(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 September 2003 (04.09.2003)

PCT

(10) International Publication Number
WO 03/073390 A2

(51) International Patent Classification[7]: G07F 7/10

(21) International Application Number: PCT/IB03/00763

(22) International Filing Date: 26 February 2003 (26.02.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
02/02570 28 February 2002 (28.02.2002) FR

(71) Applicant *(for all designated States except US)*: **SCHLUMBERGER SYSTEMES** [FR/FR]; 50 Avenue Jean Jaures, F-92120 Montrouge (FR).

(71) Applicant *(for MC only)*: **SCHLUMBERGER MALCO INC** [US/US]; 9800 Reistertown Road, Owings Mills, MD 21117 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **FAMBON, Olivier** [FR/FR]; 2 rue du Commandant Gillot, F-38000 Grenoble (FR). **FREYSSINET, André** [FR/FR]; Le Sorbier, F-38760 Saint Paul de Varces (FR). **LACOURTE, Serge** [FR/FR]; Le Soleil Levant, 2 bld des Anciens d'Algérie, F-38580 Allevars les Bains (FR).

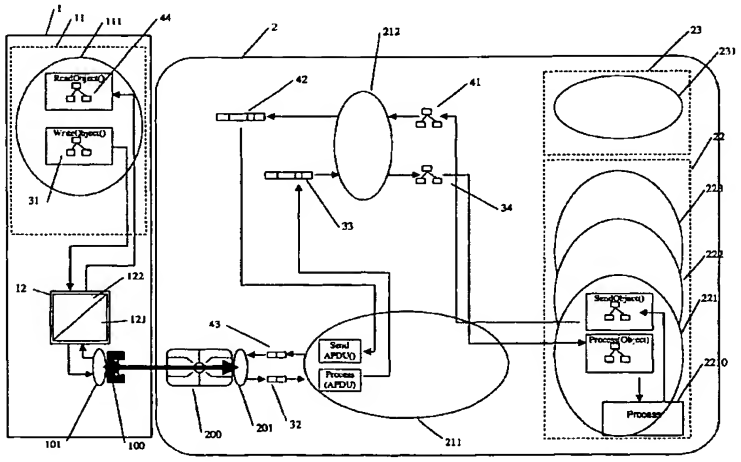
(74) Common Representative: **CORLU, Bernard**; Schlumberger Systemes, 36-38 Rue de la Princesse, BP 45, F-78431 Louveciennes (FR).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, SE, SI,

[Continued on next page]

(54) Title: ITERATIVE SERIALISATION PROCEDURE FOR STRUCTURED SOFTWARE OBJECTS



(57) Abstract: This invention concerns an iterative procedure for conversion of structured software objects into a raw data stream and vice versa, providing for their direct transfer using simple communication resources such as those of an embedded computer station, and reset of said software objects or reutilisation of memory space allocated to them.This procedure can be used by an embedded platform (2) or a portable object including at least a processor capable of exchanging information with a terminal in the form of linear data sequences. The procedure includes a step for conversion of a data set, in one direction or the other, between a linear data sequence arrangement on the one hand, and a structured arrangement describing or representing an object-oriented software object on the other hand.

WO 03/073390 A2

SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Iterative serialisation procedure for structured software objects

This invention concerns an iterative procedure for conversion of structured software objects into a raw data flow and vice versa, providing for their direct transfer using simple communication facilities such as those of an embedded computer station, and reset of said software objects or reutilisation of memory space allocated to them.

Within an application programmed in a high-level language, and in particular in structured or object-oriented languages such as Java®, a large proportion of the data used or processed is organised and stored in the form of software objects with a precise structure, more complex than a simple sequence of bytes. Such objects can contain a number of variables or data groups, for example in the form of simple types, such as characters ("char" type) or integers ("int" type), or basic object types or types defined by the programmer, character strings ("string" type), or arrays ("array" type) with one or more dimensions of the types defined above. Structured objects of this type can therefore be defined or represented by a tree structure grouping various objects themselves of varying types, the organisation of a structure of this kind sometimes being referred to as the object composition graph. An object type can be defined "by inheritance" from the definition of another type (generally referred to as a "super-type"), and all types can be represented in the form of a tree structure referred to as the inheritance graph.

According to the particular case, it can be useful to be able to transfer, create or delete software objects handled by a computer

application, in hardware or software environments where these functions are not always possible or practical.

For example, numerous computer applications incorporate communication between a computer terminal and possibly a computer network on the one hand, and a portable object or embedded platform with data processing capabilities, such as a bank, telephone subscription or health care management smart card, or a computerised identification, marking or toll device on the other.

Such portable objects incorporate, in particular, a processor associated with storage devices, containing at least one application program and with facilities for communication with one or more terminals. These communication facilities are based for example on a transmission of electronic data using different types of technology, such as electrical contacts, radio antenna, luminous or other forms of transmission, it being possible to combine a number of different types of transmission in the same portable object. For dimensional and in some cases historical reasons, the communication facilities regularly employed operate with simple protocols such as the APDU protocol to standard ISO 7816 for smart cards. Some of these protocols only provide for the transfer of simple type objects (in the form of integers or simple characters) as control parameters transmitted to or received in response from the object, or only at the initiative of the terminal or both. In the case of the ISO 7816 standards, the APDU protocol only allows transfer of objects without types in raw data form, as bytes sent to the computerised portable object as control parameters, or obtained in response from the object, and only at the initiative of the terminal.

In the current state of their evolution, some of these portable data processing objects such as JavaCard® incorporate internal functions, enabling them to exploit programmed application elements

directly in high-level or object-oriented languages such as Java®, and to be integrated in centralised or distributed applications communicating on an extended basis. To communicate with applications located inside an embedded platform of this type, the programmer of such an application must avoid using structured software objects, or provide for direct processing, and on a case-by-case basis, of the transfer of these objects to and from this embedded platform.

To take full advantage of the performance and the flexibility of a language of this type when programming an application which can be executed in the processor of a portable object of this type, it is consequently useful for the application to be able to communicate easily with other applications external to the object. It is therefore useful in this context, for the application to be able to exchange the objects it processes, without any loss of organisation or structure for these objects, with said external applications.

In programming languages such as Java®, the procedures used for serialisation/deserialisation of structured objects can rely on substantial hardware and software resources. These procedures are used in particular to save a structured object to a file, or transmit a structured object between two program processes executed in two different working memory fields.

These procedures nevertheless require software and hardware resources which are not available in certain embedded platforms, including Javacard® in particular. For example, the base classes used by a conventional Java platform occupy more than 1 MB of memory space (Java Developpement Kit: 9 MB), while a standard Javacard smart card only has 16 kB memory capacity.

Conventional serialisation procedures use the conventional resources of the Java environment, and their algorithms are designed

with the aims of easy utilisation and computer-managed maintenance. They are consequently much more memory-greedy and much too complex for transposition to an embedded platform with such limited performance in terms of memory capacity and processor power and speed.

Furthermore, with Java for example, the serialisation procedures use a generic object type manager, implemented in the Java virtual machine (VM) executed on each hardware platform.

Due to the degree of concision necessary for adaptation to an embedded platform, the current versions of the Javacard environment do not incorporate type managers. A serialisation procedure such as that used in a standard Java configuration would not therefore have access to information representing the structure of the structured objects to be reconstituted from data received in APDU format.

Furthermore, in a conventional computer environment, memory facilities are such that the serialisation procedures implemented in object-oriented programming languages, such as Java, are not concerned with the size of the structured objects to be transmitted. The objects to be transmitted are serialised by the sender independently from their reception by the addressee, and the objects received can be deserialised by the addressee independently from the sender or the time of their transmission. Linear data streams are then transmitted and managed between sender and addressee by software mechanisms exterior to the programming environment. These streams can thus transit via high capacity buffers, and are managed by other software layers, for example by a protocol such as TCP/IP.

In the case of an embedded platform, the software mechanisms used to enable data exchange with the exterior do not

provide stream management functions ensuring the integrity and continuity of the data transmitted. Nor do the modest memory resources of an embedded platform make it possible to store linear streams, and consequently large objects, during transmission and conversions.

One of the objectives of this invention is consequently to propose a procedure enabling the programmer of an application using an embedded platform, to have access to automated software tools enabling a software agent, or application element, stored or executed in a portable object of this type, to receive from or send to another agent, situated in another computer station, data organised in the form of structured software objects, in a case where the communication facilities between sender and receiver do not allow transfer of the structures composing said objects as such, but only to transfer data in a simpler form, and where the software and hardware resources of this embedded platform are not sufficient for enabling the use of a conventional serialisation procedure.

This objective is achieved by means of a data conversion procedure which can be used by a computer station, or embedded platform, comprising a portable object incorporating at least a processor, storage facilities, and communication facilities capable of exchanging information with a terminal in the form of one or more linear data sequences, characterised in that it incorporates a data set conversion step, in one direction or another, between a linear data sequence arrangement on the one hand, and a structured arrangement, describing or representing one or more software objects structured or hierarchised according to the criteria of an object-oriented programming language on the other.

According to a particularity of the invention, the procedure includes the following steps:

- conversion, or serialisation, of a first data set to be transmitted, incorporating or representing one or more software objects, structured or hierarchised according to the criteria of an object-oriented programming language, from a structured arrangement describing or representing this object to a linear data sequence representing this first data set;
- transmission of this linear data sequence using communication facilities, from the embedded platform to at least one host, namely a terminal or computer station connected to the terminal, or from this host to the embedded platform;
- conversion after transmission, or deserialisation, of this linear data sequence to a data set arranged in one or more structured software objects reproducing or representing the first data set.

According to a particularity of the invention, the host terminal sends information to the embedded platform using a software agent, referred to as a transmission function, said information being received in the embedded platform by a response function which can initiate processing of these data by at least one addressee software agent stored in the embedded platform, and forming part of at least one application, the procedure incorporating the following steps:

- reception by a communication agent representing the addressee agent, of a data set received by the response function for said addressee software agent, said data set being arranged in a linear data sequence;
- conversion of this data set into at least one software object, structured or hierarchised according to the criteria of an object-oriented programming language;
- transmission of this structured software object to the addressee agent, and initiation of a processing, according to said object, by said addressee agent.

According to a particularity of the invention, the procedure incorporates the following steps:

- reception by the response function from the host transmission function, of at least one data item in the form of at least one transmission parameter, and transmission of this parameter to a communication agent stored or executed in the embedded platform;
- conversion, or concatenation, by the communication agent of at least one transmission parameter, transmitted by the response agent, into a data set arranged in a linear data sequence, and storage of these data in an input stream in the embedded platform;
- conversion, or deserialisation, by a serialisation agent, stored or executed in the embedded platform, of at least part of the data stored in this input stream into a data set comprising or representing at least one structured software object;
- reception of this structured software object or its references by the addressee agent.

According to a particularity of the invention, the procedure incorporates the following steps:

- transmission of a structured software object, or its representation, from a software agent forming part of an application executed or stored in an embedded platform, to a serialisation agent executed or stored in said embedded platform;
- conversion, or serialisation, by said serialisation agent of said structured software object into a data set arranged in a linear data sequence, and storage of these data in an output stream in the embedded platform;
- conversion by a communication agent, stored or executed in the embedded platform, of at least part of the data stored in this

output stream into a set of response parameters suitable for transmission by the response function;

- transmission of these response parameters from the embedded platform to the host terminal by the response function, on its own initiative or in response to the host terminal transmission function.

According to a particularity of the invention, the linear data sequence stored in the input stream or output stream represents one or more software objects, structured or hierarchised using one or more data items referred to as tags, having one or more given values each representing a given action to be executed on deserialisation of said linear data sequence.

According to a particularity of the invention, at least one tag is defined as representing one of the following actions:

- addition of a new element to the structure of the structured object represented by the linear data sequence;
- reference to an element or object, referred to as source object, as being source of the value of all or part of an element making up the structured object;
- indication that the following data item or items represent the content of an element making up the structured object;
- indication of the absence of content for an element making up the structured object.

According to a particularity of the invention, the serialisation agent serialises a structured object, referred to as source object, into a linear data set in accordance with a procedure, referred to as serialisation procedure, processing at least one of the objects, referred to as elements, making up the structure or tree structure of this structured source object, by the following steps:

- detection by the serialisation agent of the type of an element, referred to as current object, making up the structure or tree structure of said structured object;
- storage in the output stream of a data item representing a tag indicating the addition of a new element, followed by a data item representing the type of the current object;
- storage in the output stream, following the elements already present in the stream, by a serialisation agent, of the type associated with the type of the current object:
  - either by means of a data item representing the value of all or part of the structured object,
  - or by means of a data item representing a tag indicating a reference to an object as source of the value of all or part of the structured object, said tag being followed by a data item identifying said source object.

According to a particularity of the invention, the serialisation procedure converts a structured object to the output stream, storing the type of each current object in a memory stack, referred to as type stack, the locations of which are read in reverse order to their order of storage, by successive iterations of the procedure.

According to a particularity of the invention, the serialisation agent deserialises a linear data set into at least one structured result object, by means of a procedure, referred to as deserialisation procedure, which processes each of the data items stored in the input stream, by the following steps:

- read by the serialisation agent of at least one data item stored in the input stream following the data previously processed;
- analysis of this data item and implementation of an action corresponding to this data item.

One of the objectives of the invention is also to propose a structured object transfer procedure of this type, making it possible to transfer software objects between a host and embedded platform, where the operating environment of said embedded platform does not incorporate type manager for said structured objects.

This objective is achieved by the data conversion procedure described above, characterised in that the deserialisation procedure includes the loading of an element, referred to as current object, namely assignment of a direct or indirect value to all or part of said current object, said element making up all or part of the structure of the structured result object, the termination of loading the current object initiating:

- read of a data item representing an object type stored in the next location of a memory structure, referred to as type stack, and deletion of said data item from this location;
- storage, as the type of a new current object, of the type represented by the data read from the type stack.

According to a particularity of the invention, the structured object created by the deserialisation procedure is considered as complete, and transmitted to the software agent or the application which is the addressee when the type stack is empty.

According to a particularity of the invention, the deserialisation operation corresponding to a data item representing a tag, referred to as "NEW" tag and indicating a new element, comprises the following steps:

- read of at least one subsequent data item in the input stream;
- storage of the object type represented by this subsequent data item, in a memory stack, referred to as type stack, following the types being possibly already stored in this stack.

According to a particularity of the invention, the type of stack used by the deserialisation procedure comprises a LIFO type stack, that is to say a stack where the locations are read in reverse order to the order in which they were stored.

According to a particularity of the invention, the deserialisation procedure comprises a step for creation of an element making up the structure of the structured result software object, by allocating a new memory space or reuse of a free allocation, said creation being performed by a type manager agent corresponding to the type of the element to be created.

According to a particularity of the invention, the creation step for an element making up the structure of the structured result software object occurs between the read step of a data item indicating the creation of this element, and the first loading step for said element.

According to a particularity of the invention, the action corresponding to a data item, referred to as simple data item, namely a data item not representing a tag, includes a step for storage of the value of this data item in the memory space allocated to the current object.

According to a particularity of the invention, during the deserialisation procedure, an object index is attributed to at least one element making up the structure of the structured result object, said object index identifying said element uniquely, and making it possible for the element to be designated or referenced by other objects or elements.

According to a particularity of the invention, the action corresponding to a data item representing a tag, referred to as "REF" tag indicating a reference, comprises the following steps:

- read of at least one subsequent data item in the input stream;

- storage in the memory space allocated to the current object, following the data items already stored, of a data item designating an object as source of the value of all or part of the current object, said source object or element being identified by said subsequent data item.

According to a particularity of the invention, loading of the current object is considered to be terminated when the data stored in the memory space allocated to said object corresponds to a given length, said length being read from card memory by a type manager agent, or a manager agent for the types stored in the embedded platform.

According to a particularity of the invention, transmission of a linear data sequence of a given length by the embedded platform to the host, is performed by an iterative procedure, referred to as data read procedure, comprising the following recursive steps:

- transmission by the host of a communication command including at least one transmission parameter representing the length of the data already received;
- reception by the embedded platform of the transmission parameter, and comparison with the linear data sequence to be transmitted;
- response to the communication command by transmission of at least one return parameter representing the data item or items following immediately after the data already received by the host;
- reception by the host of the communication command return parameter, and storage of the data which this represents following the data already received.

According to a particularity of the invention, reception by the embedded platform of a linear data sequence of a given length from

the host, is executed following an iterative procedure, referred to as data write procedure, comprising the following recursive steps:

- transmission by the host of a communication command with at least a first transmission parameter representing the data item or items following immediately after that part of the linear data sequence to be transmitted which has already been sent and, where appropriate, a second transmission parameter representing the position in the sequence to be transmitted, of the data represented by the first transmission parameter or the length of the data already transmitted;
- reception by the embedded platform of the communication command transmission parameter or parameters;
- storage in an input stream in the embedded platform of the data represented by the first transmission parameter, following the data already received.

According to a particularity of the invention, the data write procedure also comprises the following steps:

- comparison by the embedded platform of the second transmission parameter, and comparison with the length of the data already received;
- return by the embedded platform of at least one response parameter, representing either the length of the data already received, or a data item representing the result of this comparison, or both.

One of the objectives of the invention is also to propose a structured object transfer procedure of this type, making it possible to transfer software objects of a substantial size in relation to the possibilities of the communication facilities in terms of transfer or temporary storage, or software objects of unlimited size.

This objective is achieved by the data conversion procedure described above, characterised in that at least two serialisation, deserialisation, data read or data write procedures are executed in parallel or interleaved by repeating a step comprising successive execution of at least one step of each of these procedures.

According to a particularity of the invention, the input stream, output stream or both are stored in the form of circular memory structures, the two streams sharing possibly the same circular structure.

According to a particularity of the invention, the embedded platform has a programmable environment capable of storing and executing at least one application created by a programmer, the communication function being compatible with the APDU format defined in standard ISO 7816.

According to a particularity of the invention, the deserialisation and consequent processing operations for an object received are initiated on reception of at least one APDU format command containing at least one data item indicating reception of a structured object.

According to a particularity of the invention, the embedded platform has a programmable environment compatible with the JavaCard® standard.

According to a particularity of the invention, at least one of the applications executed on the host or embedded platform is programmed in Java® language.

According to a particularity of the invention, the procedure is used for communication between at least one software agent, referred to as card agent, stored or executed in the embedded platform, and at least one software agent, referred to as card engine proxy agent, stored or executed in at least one host belonging to a computer

network communicating by asynchronous messages using an AAA-MOM type software infrastructure, said card engine proxy agent serving as an intermediary for the card agent for its communications with other agents in said network, said agents operating in accordance with the specifications of said software infrastructure and belonging to at least one distributed application.

Furthermore, in certain environments or operating systems, for example of the embedded type such as JavaCard®, there is no procedure, or only complex procedures which are costly in terms of resources, for deleting a software object of this type when it is no longer used, or to free the memory space which it occupies. Manual deletion is consequently required in this case, and must be provided for directly and on a case-by-case basis by the application programmer.

One of the objectives of this invention is consequently to propose a procedure which provides the programmer of an embedded platform-based application, with software tools allowing reutilisation of memory space occupied by all or part of certain structured software objects used by an application in a data processing station, whether portable or not.

It can also be useful to have software tools for simple duplication of a software object with a non-linear structure, for example in tree structure form.

One of the objectives of this invention is consequently to propose a procedure providing the programmer of an embedded platform-based application with software tools for duplication of a structured software object, referred to as source object, to another object, referred to as result object, containing the same values as the source object but constituting a different object in storage.

Likewise, for reasons of security, when memory space occupied by a software object of this type is freed, it can be important to erase information from this object from said space, to prevent said information being read by another object or another application using the same memory space subsequently. This information must therefore be erased manually, provision for this function being made directly and on a case-by-case basis by the application programmer.

One of the objectives of this invention is therefore to propose a procedure providing the programmer of an embedded platform-based application with software tools for erasing or uniformising information contained in the memory space used by a software object of this type, on deletion of said object or reutilisation of said space.

The invention consequently proposes a procedure as described above, characterised in that the data read, data write, deserialisation and serialisation procedures are applied via their implementation in at last one class stored in the host or embedded platform, said implementation involving at least one of the following commands:

- an object write command, executing transmission of a structured object to at least one agent of the embedded platform, by utilisation of the serialisation procedure for this object, followed by the data write procedure and the deserialisation procedure;
- an object read command, executing read of a structured object from at least one agent of the embedded platform, by utilisation of the serialisation procedure, followed by the data read procedure and the deserialisation procedure;
- a deallocation command for a structured object stored in the embedded platform, by utilisation of the serialisation procedure, the latter also including a step for storage of freeing or de-allocation of the memory space allocated to each component of said object, following analysis of the structure of said component;

- a housekeeping or erase command for a memory space freed by a structured object, by utilisation of the deserialisation procedure to create an object with non-material content from a given linear data sequence;
- a duplication command for a structured source object by utilisation of the serialisation procedure, without deallocation of said source object, to create a linear data sequence representing this object, followed by utilisation of the deserialisation procedure from this linear data sequence, to create another structured object the content of which is identical to that of the source object.

According to a particularity of the invention, the programming language for the embedded platform comprises a first class (IOApplet) describing an ProcessAPDU abstract method initiating a process which can be user-defined in the application on reception of an APDU message; the program code executing the deserialisation operations in the embedded platform being stored in the embedded platform, for implementation of this ProcessAPDU abstract method, in a second class (ObjectIOApplet) inheriting from the first class (IOApplet), said program code using a ProcessObject method, the latter being described as an abstract method in this same implementation class (ObjectIOApplet).

According to a particularity of the invention, the programming language for the embedded platform includes a first class (IOApplet), describing a SendAPDU method transmitting an APDU format message to the host; the program code executing serialisation operations in the embedded platform being stored, for implementation of at least one SendObject method using the SendAPDU method, in a second card class (ObjectIOApplet) inheriting from the first class (IOApplet).

Another objective of the invention is to propose a computer system including a portable object and software tools of the types described above.

This objective is achieved with a computer system comprising a computer station, referred to as embedded platform, comprising a portable object incorporating at least a processor, storage facilities and communication facilities capable of exchanging information with a terminal in the form of one or more linear data sequences, characterised in that the platform incorporates a serialisation agent capable of executing a conversion step for a data set in one direction or the other, between a linear data sequence arrangement on the one hand, and a structured arrangement describing or representing one or more software objects, structured or hierarchised in accordance with the criteria of an object-oriented programming language, on the other.

According to a particularity of the invention, the embedded platform includes a communication agent capable of:

- receiving, in place of the addressee agent, a data set received by the response function for an addressee software agent stored in the embedded platform, said data set being arranged in one or more linear data sequences;
- converting this data set into at least one software object, structured or hierarchised in accordance with the criteria of an object-oriented programming language; and
- transmitting this structured software object to the addressee agent, and initiating processing by the addressee agent according to this object.

According to a particularity of the invention, the linear data sequence representing a structured software object is stored in the embedded platform in an input or output stream, said embedded

platform incorporating a software agent, referred to as serialisation agent, capable of creating the structured object or objects represented by the input flow in the embedded platform, namely deserialising said structured objects, or writing data representing the structured object or objects to be transmitted to the output stream, namely serialising said structured objects.

According to a particularity of the invention, the input or output stream is stored in the form of one or more circular memory structures.

According to a particularity of the invention, the serialisation agent uses a memory stack, referred to as type stack, to store the type of at least one object making up all or part of the structure of a structured object to be serialised or deserialised, said type stack having a number of memory locations which can only be accessed after the memory locations loaded most recently have been read and erased.

According to a particularity of the invention, the data contained in the input or output stream represents one or more structured objects, using a coding system comprising a set of tags, each of said tags representing a given action to be executed on deserialisation of the linear data sequence.

According to a particularity of the invention, at least one tag is defined as representing one of the following actions:

- addition of a new element to the structured object structure represented by the linear data sequence;
- referencing an element or object, referred to as source object, as the source of the value of all or part of an element making up the structured object;
- indication that the following data item or items represent the content of an element making up the structured object;

- indication of the absence of content for an element making up the structured object.

According to a particularity of the invention, the embedded platform comprises a portable object operating in accordance with standard ISO 7816 and using APDU format commands.

According to a particularity of the invention, at least one agent or application stored in the embedded platform is programmed in Java® language, said embedded platform having a computer environment in accordance with the JavaCard® standard.

According to a particularity of the invention, the system incorporates, in the host, embedded platform or both, at least one software class implementing at least one of the following commands:

- an object write command, executing transmission of a structured object to at least one agent of the card, by serialisation of this structured object into a data stream in the host, followed by transmission of this data stream to the embedded platform, and deserialisation of said data stream into a structured object in the embedded platform;
- an object read command, executing read of a structured object from at least one agent of the card, by serialisation of this structured object into a data stream in the embedded platform, followed by reception of this data stream from the embedded platform and deserialisation of said data stream into a structured object in the host;
- a deallocation command for a structured object stored in the embedded platform, by serialisation of said object in accordance with an option comprising freeing or de-allocation of the memory space allocated to each component of this object, following analysis of the structure of said component;

- a housekeeping or erase command for a memory space freed by a structured object in the embedded platform, by deserialisation of a given linear data sequence to create an object with non-material content;
- a command for duplication in the embedded platform of a structured object, referred to as source object, by serialisation to a linear data sequence representing the same object, without deallocation of said source object, followed by deserialisation from this linear data sequence into another structured object the content of which is identical to that of the source object.

According to a particularity of the invention, the embedded platform communicates at least with a host belonging to a computer network communicating by asynchronous messages in accordance with an AAA-MOM type software infrastructure, said host incorporating a software agent, referred to as card engine proxy agent, capable of managing communications between said embedded platform and other agents of said network, said agents operating in accordance with the specifications of this software infrastructure, and belonging to at least one distributed application.

The invention and its characteristics and advantages will be understood more clearly by reading the following description, with reference to the appended diagrams where:

- Figure 1 is a partial diagram showing object transfer and conversion for read and write operations of a host or terminal to and from a software agent in an embedded platform, in a version of the invention where the card incorporates one application only;
- Figure 2 is a more detailed diagram of object transfer and conversion for read and write

operations of a host or terminal to and from a software agent in an embedded platform, in a version of the invention where the card has one application only;

- Figure 3 is a partial diagram showing object transfer and conversion for read and write operations of a host or terminal to and from a software agent in an embedded platform, in a version of the invention where the card incorporates a number of applications;
- Figure 4 is a partial diagram showing the objects and agents involved in deserialisation of a structured software object from a data stream;
- Figure 5 is a partial diagram showing the objects and agents involved in serialisation of a structured software object to a data stream.

In certain literature (for example "Java embarqué" - Eyrolles – Paris 1999), the term "embedded computer" is used to designate a computer which is not visible per se, as it is integrated in an item of equipment possessing another function. The French term "ordinateur embarqué" is an approximate translation of the English "embedded computer".

This characteristic, namely provision of a particular function within another device, largely explains the fact that such embedded computers frequently have only limited hardware or software resources, and frequently employ a rudimentary operating system or software environment. While enhancement is predictable, the resources of computers of this type can be typically of the order of 512 kb static storage down to a much lower figure, for a 8-bit or 16-

bit processor. In the case of a smart card, available RAM capacity can be of the order of 4 kb, and currently up to 32 kb for the most powerful models.

In the general IT context, the term "platform" refers to a data processing device incorporating at least a processor and storage facilities. By extension of the above definition, and for purposes of description in this context, the term "embedded platform" is used to designate a portable object incorporating a platform of this type, said object being genuinely portable or of small size, or possessing very limited processing or storage capabilities.

The following description explains the procedure according to the invention for versions with given task and operation distribution between the various agents and applications concerned. The flexible organisation of a computer application naturally means that this distribution can be presented differently, in particular where distinctions between the various agents and their designations are abstract notions having no impact on their main operating characteristics. It is therefore obvious that the procedure according to the invention can also be implemented in other versions not described here, without deviating from the basic purport of the invention, in particular by combining the diverse variants presented for each task or agent in different ways. Likewise, abstract distribution of tasks between agents or applications stored in the same place at the same time, can also be combined in diverse variant versions not described here, without deviating from the basic purport of the invention.

The procedure according to the invention is illustrated, in the following description, principally in the case of an embedded platform

comprising a smart card using a JavaCard® type system environment, communicating with a terminal comprising a data processing station executing an application programmed in Java® language.

It is however obvious that the procedure according to the invention can be applied to other environments, the data transmission functions of which do not provide for transmission of structured software objects in a mode which is transparent for the programmer. Smart cards to standard ISO 7816, using another programming environment, for example "Windows for Smart Cards® ", or which can be programmed using another programming language, for example Visual Basic®, can also be concerned. This also applies to smart cards complying with another standard involving similar limits in regard to their communication functions, or portable objects or terminals using a platform of this type.

The procedure can also be applied to any portable objects incorporating an embedded data processing station, whether portable or not, such as an electronic automobile component, identification marker, cellular telephone or portable acquisition terminal, for example.

Likewise, the procedure according to the invention will be illustrated principally as involving the utilisation of an embedded platform communicating with a data processing station, referred to as terminal or host. This procedure can naturally also be used in a case where the terminal is a station forming part of any type of computer network, without deviating from the basic purport of the invention. Thus, the various functions presented as being executed by this terminal can also be distributed between a number of different devices, and this distribution can vary in time. Naturally, the

procedure described can also be described in a case where the embedded platform communicates directly or indirectly with one or more other embedded platforms, again without deviating from the basic purport of the invention. In general terms, the host or host terminal will consequently be defined as the application or station communicating with the embedded platform.

The procedure according to the invention can be applied, in particular, to communication between a smart card, for example to the JavaCard® standard, and a computer network, for example programmed in Java®, communicating by means of asynchronous messages in accordance with an AAA-MOM type software infrastructure. In this case, communications between the card and the remainder of the network are typically managed by a software agent, referred to as card agent proxy agent, serving as an intermediary for each of the software agents, referred to as card agents, stored in the card. In the card environment, these card agents are managed or coordinated or both, by a software agent referred to as card engine agent.

The procedure according to the invention then enables this card engine agent to communicate with the card engine proxy agent, exchanging structured software objects in accordance with Java language or AAA infrastructure standards. The fact of being able to exchange structured objects with the exterior thus enables the card to achieve enhanced compatibility with the AAA infrastructure, and to be seen by the other AAA agents of the network as an AAA type agent itself.

The procedure according to the invention can also be applied to communication between a smart card, for example to JavaCard®

standard, and a computer network, for example programmed in Java®, communicating by means of an object-oriented RPC (Remote Procedure Call) protocol, for example with a Java RMI (Remote Method Invocation) or CORBA® type software infrastructure.

Where an application requires transfer of software objects from a terminal to an embedded platform comprising a smart card, transfer often takes the form of a master/slave type communication. This means that the terminal takes the initiative for executing a transmission function to the embedded platform. To communicate, this function sends a communication command to the platform accompanied by transmission parameters. This command can then initiate one or more processing operations in the platform, and receive return or response parameters from the platform.

In the case of a smart card operating in accordance with standard ISO 7816, the parameters of this command are transmitted in an APDU (Application Protocol Data Unit) type format, comprising a data sequence organised as follows:

Transmission parameters sent with the command:

Header				Body		
CLA	INS	P1	P2	Lc	Data T	Le

CLA: one byte designating the addressee class or application.

INS: one byte designating an instruction to be executed.

P1, P2: two bytes providing information concerning the APDU command.

Lc: length of data transmitted.

Data: data transmitted.

Le: length of return data awaited.

Response parameters returned by the card:

Body	End	
Data R	SW1	SW2

Data: response data.

SW1, SW2: two bytes constituting check messages or codes sent by the card.

It will be seen that this APDU format does not provide for transmission or reception of data in any other form than a linear data sequence, namely a simple byte string. When an application is programmed in a language, or for an embedded platform for which the system does not have any other command than APDU communication commands, the programmer wishing to transfer more fully structured objects consequently has no simple software tools for the purpose. The programmer is then obliged to make provision in the application for conversion of structured objects of this type into a byte string, then using the APDU command to transmit the string and finally reconverting said objects in the opposite direction. This corresponds to manual serialisation, transmission and deserialisation of said objects, in other words programming of these operations in the smallest detail.

In the case of a programmer using Java® language for development of an application using JavaCard® standard embedded platforms, the tools available in JavaCard® provide for linear data transmission in APDU format using the following JavaCard® commands:

- "process(APDU)": the card receives data in APDU format, and initiates processing of the transmission parameters which they contain, by the addressee software agent or "applet" designated in these data;

- "sendAPDU()" or "APDU.sendbytes()": before or during the process previously initiated, the card returns other data in APDU format and containing return parameters to the host.

The "sendAPDU()" command is implemented in the JavaCard environment in the "IOApplet" class, in the form of a method applicable to a non-typed object containing raw data.

The "Process(APDU)" command on the other hand is declared in the "IOApplet" class in the JavaCard environment, in the form of an abstract method. This means that the method exists in the Javacard environment, but also that the code implementing the method must be written by the programmer of an application for which it is desired to use this command. For example, the programmer creates an inheriting sub-class of the "IOApplet" class in the application, this sub-class then containing a method receiving the code of the process to be executed by this "Process(APDU)" method. The JavaCard environment merely calls the "Process(APDU)" method when a message is received, and then initiates the process which the programmer has included in the additional code.

To provide a programmer in this context with the tools for direct transmission of structured software objects, the procedure according to the invention provides similar commands, but which accept structured software objects directly, in accordance with the specific object-oriented characteristics of the Java® language. These commands can then be used in the JavaCard® environment, and can be of the "process(Object)" and "sendObject()" type, for example.

To enable the programmer, also referred to here as the user, to employ these commands directly without having to become involved with APDU command syntax, the procedure according to the

invention takes over serialisation, transmission and deserialisation operations for objects and data between the application or sender agent and the addressee agent.

In the case of a JavaCard environment, these operations can be performed by executing a program code contained in the "Process(APDU)" method belonging to a sub-class of the "IOApplet" class, named "ObjectIOApplet", for example. This code is then initiated automatically by the environment when an APDU message is received, and itself contains the conversion operations and the call to another "Process(Object)" abstract method. With a version of the invention of this type, the programmer merely has to write the code for the operations to be executed by the card when a structured object is received. The programmer then writes this code in an inheriting sub-class of the "ObjectIOApplet" class, for implementation of the "Process(Object)" abstract method.

Only the conversion, serialisation and deserialisation operations executed at the card end are described in the following paragraphs. It is naturally clear that the procedure described can also be used for executing the same operations at the host end. The hardware and software resources of the host terminal being usually substantially greater than those available on the card, these operations can nevertheless also be programmed or organised differently at the host end without deviating from the basic purport of the invention.

In a version of the invention represented in Figure 1, a host terminal (1) communicates with an embedded platform, for example a Javacard® standard smart card (2). Host (1) executes at least one application (11) comprising at least one software agent (111), and

communicates with card (2) by means of an APDU format communication function (101), involving communication resources including a connection location (100), for example. This connection location incorporates a means of connection, for example by electrical contact, microwave link, IR link or magnetic track, or a combination of two or more of these types.

Card (2) executes an application (22) including at least one software agent (221), and communicates with the host (1) using a response function (201) forming part of the JavaCard® system environment. For this purpose, said response function (201) uses communication resources (200) of a type compatible with the communication resources (100) of host terminal (1).

When agent (111) of host application (11) wishes to send data to card agent (221), order it to execute a process (2210) or request information located in card storage, it executes an object write instruction initiating the procedure for sending structured software object (31) to the card, this instruction being designated "WriteObject()", for example. Said software object (31) is then serialised, namely converted to an APDU format data set by a host conversion agent (12). This host conversion agent then uses the communication function (101) to transmit these data to the card (2) via the host connection location (100) and the card communication resources (200).

When received in the card by the response function (201) which manages the communication resources (200), this response function transmits said data to a card conversion agent (21). Said card conversion agent (21) deserialises the data, converting them in the opposite direction to reobtain their original structure, in the form of a software object (34). Said structured software object (34) is then transmitted to the addressee agent (22), using an object processing

instruction which will execute the process (2210) corresponding to the data received, this instruction being designated "Process(Object)", for example.

When the agent (221) of application (22) in card (2) wishes to send data to the agent (111) of host (1), or information concerning a process executed, it executes an object transmission instruction which initiates the procedure for transmission of a structured software object (41) to the host, this instruction being designated "SendObject()", for example. Said software object (41) is then serialised, namely converted to an APDU format data set, for example by the card conversion agent (21). This card conversion agent then uses the response function (201) to send these data to the card (2) via the card communication resources (200) and the host connection location (100).

When received by the host communication function (101) which manages the connection location (100), said communication function transmits the data to the host conversion agent (12). Said host conversion agent (12) deserialises the data, converting them in the reverse direction to reobtain their original structure in the form of a software object (44). Said structured software object (44) is then transmitted to the addressee agent (11) by an object read instruction which reads the data received, this instruction being designated "ReadObject()", for example.

Figure 2 represents a version of the invention where serialisation/deserialisation and transmission operations executed by the host and card conversion agents (12 and 21 respectively), are executed by two different agents (121, 122 and 211, 212 respectively).

The card conversion agent (21) thus comprises a communication agent (211) and a serialisation agent (212). The

communication agent (211) manages the byte conversion operations, exchanging data in the form of transmission parameters (32) and response parameters (43) with the response function (201). For data reception, said communication agent (211) verifies that the data received are complete, and concatenates transmission parameters (32) in a linear data sequence stored in an input stream (33). For data transmission, said communication agent (211) reads a linear data sequence in an output flow (42), and separates these data for distribution as response parameters (43) compatible with the transmission capabilities of the response function (201).

The serialisation agent (212) manages the object level conversion operations, performing serialisation/deserialisation proper. For data transmission and consequently serialisation, the serialisation agent (212) analyses the structure and content of a structured software object (41) to be transmitted, and codes this object in the form of a linear data sequence stored in the output stream (42). For data reception and consequently deserialisation, the serialisation agent (212) reads a linear data sequence in the input stream (33). It then analyses the data in this stream, and reconstitutes the structured software object (34) which they represent.

Figure 3 represents a version of the invention where the card can have a number of agents (221, 222, 223, 231) which can have addressee status for the procedure according to the invention, these agents being possibly distributed between one or more applications (22, 23). To be able to handle all data exchanges between the card (2) and the host (1), the procedure according to the invention includes an interface agent (210), via which all structured data exchanges between the card (2) and the host (1) transit.

In this version of the invention, the interface agent (210) receives all data transmitted to the card, in place of the application (22) or addressee agent (221), irrespective of said addressee. Said interface agent (210) then directs concatenation of transmission parameters (32) by the communication agent (211), and subsequent deserialisation of resultant data (33) by the serialisation agent (212), as described above. Once these data have been reconstituted as a structured software object (34), the same interface agent (210) transmits said structured object (34) to the software agent (221) which was the addressee for data (32) on their reception. In another version (not shown), the data (32) received by the interface agent (210) contain identification information for the addressee agent (221), or the interface agent (210) adds corresponding information to said data (32) before these are transmitted. The structured object (34) resulting from deserialisation is then addressed directly by the deserialisation agent (212) or a distribution agent (not shown).

In the same way, the interface agent (210) receives all data (41) transmitted to the host by a sender application (22) or a sender agent (221). Said interface agent (210) then directs serialisation of these data, by the serialisation agent (212), into an output stream (42), and concatenation of said output stream (42) by the communication agent (211), as described above. The data obtained are then sent by the response function (201) to the terminal (1) via the communication resources (200, 100) in the form of response parameters (43).

In another version of the invention (not shown), the interface agent (210) is inserted in the transmission and conversion chain (201, 211, 212, 221). For reception in the card (host write command), the interface agent (210) directs the data or objects received to their addressee (221, 222, 223, 231) in the card, according to information

included in the data received. For transmission from the card (host read command), the interface agent (210) can also be included to receive data or objects to be transmitted, assigning information to said data or objects representing their sender (221, 222, 223, 231).

It can be considered that information exchanges between the host (1), whether this is the terminal itself or any software agent capable of managing this terminal, and an agent or application present on the card, are consequently executed at two different levels, namely byte level and object level.

At object level, the data organised in structured software objects to be transmitted are converted into a linear data stream, and vice versa. This serialisation step manages the structure of the object, and is executed by the serialisation agent in each platform.

At byte or byte string level, the data arranged in simple linear streams are transmitted by fragment using the transmission function, for example in APDU format, this being the only means of exchanging information between the card and the outside world. These exchanges are managed by the host and card communication agents, which communicate with each other by sending parameters using this transmission function.

To provide the user with transparent commands, the various processes are implemented in one or more classes, and generally with at least one operating in the card and one in the terminal or host. In Java® language, the procedure according to the invention can so provide an "ObjectIOApplet" class for the card and an "ObjectIOProxy" class for the host, using the following syntax:

```
class ObjectIOProxy {
    void        writeObject (Object o);
    Object      readObject ();
```

```
}
```

with the "ObjectIOProxy" class defining a class in the host, providing the user application with "writeObject (Object)" and "readObject ()" commands:

```
class ObjectIOApplet {
    void        process (Object o);
    void        sendObject (Object o);
}
```

and with the "ObjectIOApplet" class defining a class in the card, providing the user application with "process(Object)" and "sendObject()" commands.

As the card is essentially passive, all these conversion and transmission operations are executed at the initiative of the host, using an instruction in the code which triggers transmission of an APDU command. It is this command that initiates the conversion operation requested from the agent concerned.

When executed in the card application code, the "process(Object)" and "sendObject()" commands do not therefore trigger the corresponding conversion operations directly. The "sendObject()" command stores the object to be transmitted in an output queue, for example "qout", where the objects to be serialised for transmission are introduced by an introduction command of the "qout.push(object)" type. When a serialisation operation is initiated, these objects to be transmitted are extracted from the output queue in the same order as they were introduced.

Likewise, the command extracts an object serialised by a previous operation initiated by the host, from an input queue, for example "qin". Said extraction is obtained by an extraction command of the "qin.pop()" type.

The read, write, serialisation and deserialisation operations are then managed from the host by the "ObjectIOProxy" class, using user-transparent commands. These commands can be implemented in the code of this class, in the following form for example:

- "card.Serialize out": initiates serialisation in the card of the object to be transmitted, as described below. These objects can have been stored in an output queue on execution of a "sendObject()" instruction by the card application;
- "card.Read": initiates output stream data read in the card, and transmission of these data to the host, as described below;
- "card.Write": initiates transmission of data to the card, and data write to the input stream, as described below;
- "card.Serialize In": initiates deserialisation of the objects present in the input stream in the card, as described below.

In the version of the invention described here, the card output and input streams are stored in the same circular memory structure. Other versions of the invention are naturally possible, using a different memory structure for each stream or using other types of memory structure.

A circular memory structure corresponds in this case to a set of successive memory locations, the first location being considered by the system as following immediately after the last location. This means that a linear data sequence can be stored in a structure of this type starting at any location, without loss of continuity, provided the length of said sequence is not greater than the total number of locations.

The fact that the input and output streams share the same circular structure involves storing the data which they contain in different areas of this same circular structure. The data contained in

these streams being erased as they are read, immediately or during an operation, the length and position of the two streams vary inside this circular structure. When one of the two streams has no more space to store new data as it is blocked by non-erased data of the other stream, it is merely necessary to process the other stream to free the space which it occupies.

A circular structure of this type makes it possible to use limited storage resources for processing streams the length of which is not determined in advance. It is merely necessary for the different operations on the two streams to be interspersed sufficiently evenly for this purpose.

At byte level, data exchanges from a platform output stream to the input stream of the other platform, and vice versa, are executed as described below.

If the host wishes to obtain data located in the card output stream, it sends a read command, for example "card.Read" for Java® syntax. This command initiates a read session, typically divided into a number of transactions, each transaction representing one iteration of the process.

To initiate the read session, the host issues an APDU format command, accompanied by transmission parameters (P1 and P2, corresponding to a short integer) representing the length of the data which it wishes to receive, via its communication agent and transmission function. On receipt of this command from the communication agent, the card reads a first data block in the card output stream, and transmits this block to the card response function. This first data block is then returned to the host as response data (DataR) in the response to the APDU command.

Each successive iteration of the read process then involves transmission of an APDU command by the host, accompanied by transmission parameters (P1 and P2, namely a short integer) representing the length of the data already received. Once received by the card communication agent, this length serves as an acknowledgement for previous transmissions. Thus, the communication agent returns the output stream data immediately following the length indicated by the host, as response data (DataR). It should be noted that data are read from the output stream but are not erased, and can consequently be retransmitted in case of need. This means that no data can be forgotten in this transmission.

The read session terminates when the length requested by the host has been received correctly, and the host stops the iterations. The session can also be terminated, or interrupted, if the card communication agent sends a data item (for example, a DataR field with zero length) or a code (SW1 or SW2) signifying that all available data in the output stream has already been sent. If the card output stream is empty, the host must then initiate a new object serialisation operation from the card "qout" queue to the card output stream, in the card, for the output stream to receive fresh data.

When the host wishes to transmit data from its output stream to the card, it sends a write command, for example "card.Write" for Java® syntax. This command initiates a write session, typically divided into a number of transactions, with each transaction representing one process iteration.

To initiate the write session, the host sends an APDU format command, accompanied by transmission parameters (P1 and P2, namely a short integer) representing the length of the data it wishes to transmit, via its communication agent and transmission function.

The data field (DataT) of this first APDU command then contains the first data block or group to be transmitted, this data block being read in the host output stream. On receipt of this command, the card response function transmits said first data block to the card communication agent. The card communication agent then writes this first data block to the card input stream.

Each successive iteration of the write process involves transmission of an APDU command by the host, accompanied by transmission parameters (P1 and P2, namely a short integer) representing the length of the data already transmitted. The data field (DataT) of this first APDU command then contains the next data block or group read from the host output stream. On receipt of this command, the card response function transmits these parameters and this data block to the card communication agent. The card communication agent then compares the data length announced in the transmission parameters (P1 and P2) with the length of the data already received since the beginning of the write session. This means that no data can be forgotten in this transmission.

If this comparison identifies no errors, the card communication agent writes this data block to the card input stream. If an error is detected, the communication agent returns a code or index to the host, indicating an error and/or representing the nature of this error. This code or index can be returned via the response parameters (SW1 and SW2) of the response function, the returned data field (DataR), the length of this field or a combination of these elements.

The write session terminates when the length announced by the host has been transmitted, and the host stops the iterations. The session can also be terminated, or interrupted, if the card communication agent sends a code or index indicating that the card

input stream cannot receive any fresh data. The host must then initiate one or more operations in the card for freeing space in the memory structure containing this input stream.

This can involve initiating in the card a new data deserialisation operation from the input stream to the "qin" input queue of objects accessible to the card application. This can also involve initiating a new read session to receive data contained in the card output stream, and thus free space in the circular structure containing the two streams.

At object level, serialisation and deserialisation operations are executed between the data streams and agents or applications of the card, and vice versa, as follows.

Objects are deserialised in the card from input flow (33) to the "qin" input queue, where they are directly accessible to the "Process(Object)" instruction, provided by the "ObjectIOApplet" class and used by the addressee agent or application.

For implementation of the procedure according to the invention, the host uses an instruction when it wishes to initiate a deserialisation operation in the card, for example "card.Serialize In" for Java® syntax.

Figure 4 illustrates deserialisation of a structured software object (34) by decoding the data of input stream (32) corresponding to said object (34), and storage and structure component creation operations for said structured object (34), or result object.

Reading one after the other the data stored in a linear sequence in the input stream (33), the serialisation agent (212) interprets these data according to a given code, and creates a structured software object (34) based on this interpretation. Certain

data in the data sequence stored in this input stream can have a given value, and this value is interpreted as indicating the presence of a code tag. This decoding can be performed by calling an allocation method particular to each type of structured object to be decoded, for example using the following syntax:

Typeobject::decode (Object, InputStream)

to decode an object "Object" of type "Typeobject", from the input stream "Input Stream".

This code comprises a set of tags each having a given significance, and each corresponding to at least one specific value of a data item in the input stream. Thus, each data item in the input stream having a value corresponding to one of these tags is interpreted by the serialisation agent during the deserialisation process.

For applications using the procedure according to the invention, the serialisation agent (212) can be designed to recognise a number of tag sets, or each tag can be represented by a number of different data values, without deviating from the basic purport of the invention. Diversity of this type can be used, in particular, for accounting for one card (2) or card type, with a number of different terminals or host environments.

For the version of the invention described here, the code comprises tags of the "NULL", "NEW", "REF" and "DATA" type.

- A NULL tag represents an absence of data, while occupying a location in the structure in course of construction by the serialisation agent.
- A NEW tag indicates the beginning of the description of a new object to the serialisation agent, this object being structured software object (34) resulting from the conversion, or an object,

referred to as element, comprising part of a structured object (34) of this type.

- A REF tag indicates the designation of an object, the same or another object, as value source for all or part of the object or element in course of description. This is used to assign a value by reference.
- A DATA tag indicates that the following data represent the value or content of the object or element in course of description. This content can comprise raw data representing one or more values to be assigned to the object directly, or include other tags again indicating objects or references defining this content.

According to its type, a tag can be followed by one or more data items, interpretation of which is determined by the meaning of the tag concerned.

- On reading a NEW tag, the serialisation agent knows that it must interpret the following data item as representing the type of the new object described.
- Likewise, on reading a REF tag, the serialisation agent knows that it must interpret the following data item as representing an identifier for the object designated as source of this value by reference.

In the version of the invention described here, the tags are coded on one byte in the same way as type and reference identifiers but it is obvious that the procedure according to the invention also provides for the utilisation of other codes, which are different or more complex or explicit according to the needs and possibilities of the computer environment concerned.

As it reads the data of input stream (32), the serialisation agent (212) analyses the value of each data item, and creates and loads objects or elements (340, 341, 342, 343, 344) making up the

result object (34). This read of input stream (32) is repeated until reconstruction of the object (34) represented by this input stream has been completed. This creation can be obtained by calling an allocation method particular to each object type, for example using the following syntax:

Typeobject::malloc()

for creation or allocation of a "Typeobject" type object on deserialisation.

As objects of this type can have different structures, these creation and loading operations are managed, whether directly or not, by a management agent (TM0, TM1, TM2) of a type specific to the type of object to be created, for example a "type marshaller" agent in Java® language. This type management agent is specific to one or more object types, and can be managed by a generic type manager agent (TMG), for example a "type manager" agent in Java® language. In particular, this type manager agent stores the identifiers for the various object types created during the deserialisation process. This generic type manager agent (TMG) also includes the codes and procedures, or methods, specific to these different object types and used for serialisation/deserialisation of the same objects. The type manager agent is also used to manage the list of objects and their allocations, making it possible to execute new allocations and reuse a free allocation to create a new object on decoding.

Typically, the type manager agent (TMG) includes information on all the different types of object which can be managed in the card. This information can be generated by a host, for example during a card programming phase. The information is then transmitted with classes ("applets") used by the applications, in the form of a program code ("glue code") which is added to the code of the generic type manager agent (TMG).

Using a type manager agent, and type marshaller agents as described below, the procedure according to the invention makes it possible to manage the serialisation and deserialisation of structured objects of basic or constructed type, in an embedded platform the programming environment of which, such as JavaCard for example, does not include such a type manager.

Creation of an object or element during decoding corresponds to allocation of this object, namely reservation of given memory space, and allocation of this space to the object. Some embedded environments, including Javacard® in particular, do not include software tools which can be used to free memory space previously occupied by a deleted object, such as a conventional Java® language "garbage collector" agent. On creation of an object by the serialisation agent (212), the procedure according to the invention can provide for the possibility of executing this creation with allocation of memory space to this object which had previously been occupied by another object which had become superfluous, for example of the same type as the object to be created. It is consequently possible to reuse card memory space from time to time, this frequently being a valuable asset due to card memory space limitations.

In the example illustrated in Figure 4, the input stream (32) corresponds to a structured result object (34), the structure, or graph of which can be described in Java® language in the following form:

Type1: {bool}	class B {boolean bo ;}
Type2: Type1+{int, byte, Type0}	class X extends B { int i ;

	byte by ; Y y ; }
Type0: {Type1 }	class Y { B b ;}

On reading the input stream (33), the serialisation agent reads a NEW tag (321) first. It therefore reads the following data item (322), and then stores the creation request for a type 2 object. In accordance with this NEW identifier 2 tag (321), the serialisation agent creates new object "x" (340), of class X in the example, using the type manager agent (TM2) corresponding to the same type (Type2). At the commencement of description of a result object, this new object will be the "root" object of the tree structure of the graph for this result object.

On this creation, the serialisation agent assigns an index (3402) to the newly created object (340), this index taking a value of 0 in the example. Said index (3402) can thus serve as a reference for other elements or parts of elements on construction of this result object (34), making it possible to determine whether the content of this element has been loaded or not.

The serialisation agent then stores the type and index of this object in a memory structure, referred to as type stack (TYST). This type stack is a memory stack type structure, which means that data items can be stored (pushed) one on top of the other, and a given data item can only be read and extracted (popped) when the data stored subsequently have already been extracted. Data are extracted from the stack in the reverse order to that in which they were stored (LIFO for Last In First Out).

The serialisation agent also stores the type of said new object (340) and its index, as corresponding to the object, referred to as

current object (OBJ), which will be loaded with the next data in the input stream.

The next tag is a DATA tag followed by two raw data items (324, 325), which are not tags or identifiers. These two data items are therefore stored in the current object (OBJ), namely x (340) as the value of the following elements (342, 343). These two following elements are integer type (int) designated "i", and byte type (byte) designated "by" respectively, these elements taking the values contained in these two data items (324, 325) of the input stream respectively.

The next tag is a NEW tag, followed by a data item indicating type 0. This sequence indicates that the next element of object "x" is a type 0 object. The serialisation agent will therefore assign an index (3442), 1 in the example, to said object (344), and add (push) the index and type, namely 0, for said new object (344) to the type stack (TYST). The serialisation agent will also have a type 0 object, "y" (344) of type Y in the example, created by type manager agent (TM0) corresponding to type 0.

Via the type manager agent (TM2) corresponding to the current object (OBJ), the serialisation agent knows that said current object (OBJ), namely "x", is not fully loaded. The next tag, a DATA tag followed by a raw data item (329), will consequently be assigned as value for the next element (341) of the current object, namely as the value of Boolean type element "bo", this being the last element of object "x".

Via the type manager agent (TM2) corresponding to the current object (OBJ), the serialisation agent knows that said current object (OBJ), namely "x" (340), is now fully loaded. The index (3402) of said loaded object (340) is then stored in the object stack (OBJST),

with the identifier of this object (340) in the card. The serialisation agent then terminates loading of the current object, and extracts (pops) the type and index stored at the top of the type stack (TYST). The top of the stack must naturally be understood as the stack location accessible first. The type and index extracted from the type stack, 0 and 1 respectively in the example, are then stored as corresponding to a new current object (OBJ).

The next tag is a DATA tag, which therefore indicates the loading of the current object, namely "y". This tag is followed by a REF tag (3211) and a data item (3312), 0 in the example. The serialisation agent therefore assigns a value by reference to object "y" (344), representing a simple link with the value of the object for which the index (3402) is designated by this reference (3212). Object "y" will therefore have a value defined as referencing object "x", which has index 0 in the deserialisation process. This link can then be defined in the result object (34) by storing the identifier for this object "x" in the card, this identifier being read from the object stack (OBJST) by means of this index.

Via type manager agent (TM0) corresponding to the current object (OBJ), the serialisation agent knows that said current object (OBJ), namely "y", is now fully loaded. The index (3442) of said loaded object (344) is therefore stored in the object stack (OBJST), with the identifier for said object (344) in the card. The serialisation agent then terminates this loading and interrogates the type stack (TYST).

As the type stack (TYST) is empty after extraction of the type of object "y", namely there are no more "constructed types" to be reconstituted, the serialisation agent concludes that the result object (34) has been created in full.

In a variant version of the invention, the creation or allocation of a new object can be performed at any stage of the deserialisation process, from read of the NEW tag indicating said new object, up to commencement of loading of this object.

In another variant, the index used on deserialisation of an object is identical to the identifier of the object in the card.

Objects are serialised in the card from output stream (42) to output queue "qout", which can be accessed directly by the instruction " SendObject()" supplied by the "ObjectIOApplet" class and used by the addressee agent or application.

For implementation of the procedure according to the invention, the host uses an instruction, for example "card.Serialize Out" for Java® syntax, to initiate a serialisation operation in the card.

Figure 5 illustrates the serialisation of a structured software object (41), referred to as source object, by analysis of the components of the structure of said structured object (41), followed by coding in the form of data stored in the output stream (42) corresponding to said source object (41).

This serialisation function is implemented in a method, namely an action or procedure available for an object of a given type, using the following Java® syntax:

Typeobject:: (Object, OutputStream)

being a method used to code an object "Object" of type "Typeobject" to the output stream "OutputStream".

Typeobject::getSuper()

being a method used to obtain information concerning the type and constructed types of an object, on coding a "Typeobject" type object.

The basic types are generally those types scheduled and managed by the environment or programming language, in contrast to constructed types, defined as a combination of a number of objects. Current basic types are "integer", "Boolean" and "byte" for example for an embedded environment, and long integer, real, long real and character types ("long", "real", "double" and "char") for more comprehensive environments.

For this serialisation operation, the serialisation agent (212) executes a recursive scan on the complete structure, or graph, of the source object (42) to be serialised, analysing the object elements (410, 412, 413, 414, 411), starting with the "root" object or element, "x" in the example. The graph of the source object (42) presented in the example is the same as described above for Figure 4. The serialisation agent calls an agent (TM0, TM1, TM2) corresponding to the same constructed type, for each graph element having a constructed type.

Description of the source object (41) in the output stream (42) commences by writing a NEW tag, followed by the root element type identifier, type 2 object "x" in the example. This root object is then designated as current object (OBJ). This object also receives an index, 0 in the example, and its type and index are then loaded (push operation) in the type stack (TYST).

The description is continued by writing a DATA tag, followed by data indicating the values or references corresponding to the content of the root object. As root object "x" contains an object "y" (414) which is a constructed type, the description of the root objective includes a NEW tag, followed by the type of said object "y", "NEW 0" in the example, in place of the value of said object "y". An index is assigned to the NEW tag, 1 in the example, when this tag is written.

The index and type of this new object are then loaded (push operation) in the type stack (TYST).

When description of the content of object (OBJ), namely object "x" (410), has been completed, the index of this object is stored in the object stack (OBJST) with the identifier of this object. The serialisation agent then interprets the type stack, extracting ("pop" operation) the type and index of object "y". It stores said object "y" as new current object (OBJ), and then commences description of the content of said object "y". This object comprising a simple reference to the root object "x", the data written to the output stream comprise a REF tag, followed by a data item serving as index for the object to be designated as reference, namely the object "x" with index 0 in the example.

Once description of the content of the current object, namely object "y" (414) has been completed, its index is stored in the object stack (OBJST) with the identifier of this object. The serialisation agent then interrogates the type stack, extracting (pop operation) the type of object "x" which was stored under object "y" in this type stack. As the index of object "x" is already stored in the object stack (OBJST), the serialisation agent knows that object "x" has already been serialised or described. It consequently interrogates the type stack again, observes that it is empty, and consequently concludes that all elements making up the source object (34) have been fully described in the output stream (42).

Due to the recursivity of these algorithms, it is possible to perform these serialisation and deserialisation operations using a program code which takes up sufficiently little memory space for it to be possible to store the code in an embedded platform, for example a smart card or JavaCard standard portable computerised object. The

concision of these algorithms also makes it possible to execute these operations with a low-power processor, such as those used in such embedded platforms.

In order to balance the volume of data stored in temporary form, the various read, write, serialisation and deserialisation operations required by the host application (11) can be interleaved in one or more program loops. A loop of this type executes a command for each of these operations on each iteration, for example using the following Java® syntax:

```
Do
    Do card.Serialize out While (ok out)
    Do card.Read While (data read)
    While (data in) card.Write
    While (ok in) card.Serialize in
Loop
```

In this example, the first and last lines ("Do" and "Loop") determine the repetition of a loop comprising the four intervening lines of code. A loop of this type can naturally be combined with other operations, and include various conditions for interrupting the iteration process.

The first line in the loop indicates initiation of a serialisation session to the card output stream, for objects previously stored in the "qout" output queue by a "SendObject()" command, in the card. This session is then repeated as long as an "ok out" condition is met, for example as long as there are objects to be transmitted in the "qout" queue, and the output stream is not full.

The second line indicates initiation of a read session for data contained in the card output stream by the host. This session is then

repeated as long as a "data read" condition is met, for example as long as data are received from the card.

The third line indicates execution and repetition of a write session to the data input stream from the host. This session is only executed and repeated if, and as long as a "data in" condition is met, for example as long as there are data to be sent to the card and the card entry stream is not full.

The fourth line indicates execution and repetition of a deserialisation session for data contained in the card entry stream to the structured object "qin" entry queue, where these data are extracted by a "process(Object)" command. This session is only executed and repeated if, and as long as an "ok in" condition is met, for example as long as there are data to be deserialised in the card entry stream.

In the case of a passive card, to JavaCard® standard for example, the end of an object serialisation operation typically initiates processing of the object, for example by automatic call to the "process(Object)" method.

It should be noted that only operations executed at the card end are illustrated in this example. Symmetrical operations executed at the host end can be performed either in a similar or totally different way, according to the hardware and software resources available, without deviating from the basic purport of the invention.

Due to the fact that these various complementary operations are interleaved in a software loop which can be repeated recursively, the various phases involved in the transfer of a structured object can be executed in parallel, within the framework of a single execution or single transfer command. Using data streams stored in circular form,

and reusing the same memory space indefinitely, as described above, the same command can initiate complete transfer of a structured object with no restriction on size despite the limited capacities of the embedded platform.

Where all the conversion operations described above are implemented, namely programmed, in one or more procedures loaded in the host and the embedded platform, said procedures can then be accessed by the user with a few simple commands.

In the version of the invention described here, as applied to the Java® language and JavaCard® environment, an application programmer only needs to use these few simple commands to create the application. These commands perform all intermediate operations transparently for the user. In other words, the user has no need to become involved with internal operation of the mechanism of said commands.

For example, the "ObjectIOProxy" class loaded in the processing station or host terminal supplies the "WriteObject()" and "ReadObject()" commands.

Likewise, the "ObjectIOApplet" class loaded in the embedded platform supplies the "Process(Object)" and "SendObject()" commands. Typically, the "ObjectIOApplet" class implements these commands using a known extension technique, namely by inheritance from the "IOApplet" class which contains the "process(APDU)" method. This involves adding the program code combined with the initial method to this method, and modifying or replacing operation of the code as defined in the initial method.

Instruction "WriteObject()" is employed by the user in its host application, for transmitting a structured object to the card application and initiating processing in the card.

The "Process(Object)" method is called automatically by the card following reception and deserialisation of a structured object. The content of this method is then programmed by the user in that part of the application loaded in the embedded platform, to execute the desired operations from this object. This method is then used in a similar way to the standard "process(APDU)" command in JavaCard® for APDU format data only.

The "SendObject()" instruction is employed by the user in that part of the application loaded in the embedded platform, for example inside the "Process(Object)" method code, to prepare transmission of one or more structured objects from the embedded platform to the host. This instruction is then used in a similar way to the standard "sendAPDU()" command in JavaCard® for APDU format data only.

The "ReadObject()" instruction is employed by the user in its host application, for reception of the structured object or objects which the card application has prepared for this purpose, from the embedded platform.

It will be understood that is easier for a programmer creating an application involving an embedded platform of this type, to transmit structured software objects between said embedded platform and a host or terminal, even if the communication resources of said embedded platform are not capable of transmitting data in byte or integer form.

It can be useful for the programmer of an embedded application to have a few simple commands for deallocation of a

structured software object, or reset or erase of corresponding memory space, as also duplication of a structured software object. This is particularly true in a case where the programming environment of the embedded platform does not possess a garbage collector software tool managing reutilisation of memory space already allocated, for example in the JavaCard® environment.

In a version of the procedure according to the invention, the analysis step for each component of a structured software object during serialisation of the object, can be performed using various options. A number of these options can be included in implementation of the same serialisation command, each execution of this command being accompanied by a parameter indicating which option must be used. These various options can also be used separately in different implementations of the serialisation command.

In one of these options, on analysis of each component of a structured software object during serialisation of the object, the serialisation agent (212) deallocates the memory space containing this component, or marks said component as being reusable for a new allocation. Thus, serialisation using this option, referred to as destructive serialisation, executes conversion of a structured object after which the memory space occupied by the object is freed.

In another version, the procedure according to the invention can include an option or command which executes this type of destructive serialisation of a structured object stored in the card, outside any transmission operation, for example in implementation of the "ObjectIOApplet" class. It will be clearly understood that the procedure according to the invention enables the user to reuse memory space occupied by all or part of some of the software objects of the application, without difficulty.

In another version (not shown), the procedure according to the invention includes an option or command executing deserialisation of a structured object by reusing the memory space in the card previously occupied by a structured object which has since been destroyed, as described above. This deserialisation is executed, outside any transmission operation, from a data stream containing data which is all identical or non-material. After allocating and writing an object of this type, the memory space reused does not contain any data belonging to the object previously occupying the space concerned. When this operation is implemented, for example in the "ObjectIOApplet" class, the procedure according to the invention enables the user to program total erase of memory space corresponding to a given allocation without difficulty.

In another version (not shown), the procedure according to the invention includes an option or command executing serialisation of a structured source object to a linear data sequence, outside any transmission operation. This same linear data sequence is then deserialised into a result object identical to the source object, but using a different memory space. When this operation is implemented, for example in the "ObjectIOApplet" class, the procedure according to the invention enables the user to program identical duplication of a structured software object without difficulty.

It will be obvious to the specialist in this domain that this invention provides for versions in many other specific forms, without deviating from the field of application for the invention as claimed. Consequently, the version described must be regarded as being provided for illustration purposes only, and can be modified in the

domain defined by the scope of the attached claims. The invention must not therefore be limited to the details given above.

CLAIMS

1. Procedure for data conversion for use by a computer station (2), referred to as "embedded platform", comprising a portable object incorporating at least a processor, storage facilities and communication resources capable of exchanging information with a terminal in the form of one or more linear data sequences, characterised in that it includes a conversion step for a data set, in one direction or the other, between an arrangement comprising a linear data sequence on the one hand, and a structured arrangement describing or representing one or more software objects, structured or hierarchised according to the criteria of an object-oriented programming language on the other hand.

2. Procedure according to the preceding claim, characterised in that it includes the following steps:

- conversion, or serialisation, of a first data set to be transmitted, comprising or representing one or more software objects (31, 41), structured or hierarchised according to the criteria of an object-oriented programming language, from a structured arrangement describing or representing this object, to a linear data sequence representing said first data set;

- transmission of said linear data sequence by communication resources (200 or 100), from the embedded platform (2) to at least one host (1), namely a terminal or computer station connected to the terminal, or from said host (1) to embedded platform (2);

- conversion after transmission, or deserialisation, of this linear data sequence to a data set arranged in one or more structured

software objects (34 or 44) reproducing or representing the first data set.

3. Procedure according to one of the preceding claims, characterised in that the host terminal (1) sends information to the embedded platform (2), using a software agent (101), referred to as transmission function, said information being received in the embedded platform by a response function (201), capable of initiating a process (2210) for said data by at least one addressee software agent (221), stored in the embedded platform and forming part of at least one application (21), the procedure including the following steps:

- reception by a communication agent (211), in place of addressee agent (221), of a data set (32) received by the response function (201) for said addressee software agent, said data set being arranged in a linear data sequence;
- conversion of this data set into at least one software object (34), structured or hierarchised according to the criteria of an object-oriented programming language;
- transmission of said structured software object (34) to the addressee agent (221), and initiation of a process (2210), according to said object, by said addressee agent.

4. Procedure according to one of the preceding claims, characterised in that it includes the following steps:

- reception by the response function (201) from the transmission function (101) of host (1), of at least one data item in the form of at least one transmission parameter (32), and transmission of this

parameter to a communication agent (211) stored or executed in embedded platform (2);

- conversion, or concatenation, by the communication agent (211) of at least one transmission parameter (32), transmitted by the response agent (201), into a data set arranged in a linear data sequence, and storage of these data in an input stream (33) in the embedded platform (2);

- conversion, or deserialisation, by a serialisation agent (212) stored or executed in the embedded platform (2), of at least part of the data stored in said input stream (33) to a data set comprising or representing at least one structured software object (34);

- reception of said structured software object (34) or its references by the addressee agent (221).

5. Procedure according to one of the preceding claims, characterised in that it includes the following steps:

- transmission of a structured software object (41) or its representation, from a software agent (221) forming part of an application (22) executed or stored in an embedded platform (2), to a serialisation agent (212) executed or stored in said embedded platform;

- conversion, or serialisation, by said serialisation agent (212) of said structured software object (41) to a data set arranged in a linear data sequence, and storage of these data in an output stream (42) in the embedded platform;

- conversion by a communication agent (211), stored or executed in the embedded platform (2), of at least part of the data stored in

said output stream (42), to a response parameter set (43) capable of being transmitted by the response function (201);

- transmission of said response parameters (43) from the embedded platform (2) to the host terminal (1) by the response function (201), at its own initiative or in response to the transmission function (101) of host terminal (1).

6. Procedure according to one of the preceding claims, characterised in that the linear data sequence stored in the input stream (33) or output stream (42) represents one or more software objects (31 or 41) structured or hierarchised using one or more data items, referred to as tags, with one or more given values each representing a given action to be executed on deserialisation of said linear data sequence.

7. Procedure according to one of the preceding claims, characterised in that at least one tag is defined as representing one of the following actions:

- addition of a new element to the structure of the structured object represented by the linear data sequence;

- reference to an element or object, referred to as source object, as the source of the value of all or part of an element making up the structured object;

- indication that the following data item or items represent the content of an element making up the structured object;

- indication of the absence of content for an element making up the structured object.

8. Procedure according to one of the preceding claims, characterised in that the serialisation agent (212) serialises a structured object (41), referred to as source object, into a linear data set (42), following a procedure, referred to as serialisation procedure, processing at least one of the objects (410 to 414), referred to as elements, making up the structure or tree structure of said structured source object (41), by the following steps:

- detection by the serialisation agent (212) of the type (4100), of an element (410, 414), referred to as current object, making up the structure or tree structure of said structured object (41);

- storage in the output stream (42) of a data item representing a tag indicating the addition of a new element, followed by a data item representing the current object type (TYOBJ);

- storage in the output stream, after the elements already present in the stream, by a type serialisation agent (TM0, TM1, TM2) associated with the current object type (TYOBJ):

  - either by means of a data item (424, 425, 429) representing the value of all or part (412, 413, 411) of the structured object (41),

  - or by means of a data item (4211, 4212) representing a tag (4211) indicating a reference to an object (410) as source of the value of all or part (414) of the structured object (41), this tag being followed by a data item (4212) identifying said source object (410).

9. Procedure according to one of the preceding claims, characterised in that the serialisation procedure converts a structured object (41) to the output stream (42), storing, on each

iteration, the type of each current object in a memory stack (TYST), referred to as type stack, the locations of which are read in reverse order to the order in which they were stored.

10. Procedure according to one of the preceding claims, characterised in that the serialisation agent (212) deserialises a linear data set into at least one structured result object (34), following a procedure, referred to as deserialisation procedure, processing each data item stored in the input stream (33), by the following steps:

- read by the serialisation agent of at least one data item stored in the input stream after the data previously processed;
- analysis of this data item and execution of an action corresponding to said data item.

11.Procedure according to one of the preceding claims, characterised in that the deserialisation procedure involves loading an element, referred to as current object, namely assignment of a direct or indirect value to all or part of said current object, said element making up all or part of the structure of structured result object (34), with termination of loading of the current object initiating:

- read of a data item (TYT2) representing an object type stored in the next location of a memory structure, referred to as type stack (TYST), and erase of this data item from this location;
- storage, as new current object type (TYOBJ), of the type represented by the data item (TYT2) read from the type stack.

12. Procedure according to one of the preceding claims, characterised in that the structured object (34) created by the deserialisation procedure is considered to be complete and

transmitted to the software agent (221) or application (22), which is addressee for said object, when the type stack (TYST) is empty.

13.Procedure according to one of the preceding claims, characterised in that the deserialisation action corresponding to a data item (321, 326) representing a tag, referred to as "NEW" tag, indicating a new element, involves the following steps:

- read of at least one subsequent data item (322, 327) in the input stream;
- storage of the object type, represented by said subsequent data item (322, 327) in a memory stack, referred to as type stack (TYST), following the types which may already have been stored.

14.Procedure according to one of the preceding claims, characterised in that the type stack (TYST) used by the deserialisation procedure comprises a LIFO type stack, namely where the locations are read in reverse order to that in which they were stored.

15.Procedure according to one of the preceding claims, characterised in that the deserialisation procedure includes a step for creation of an element (410 to. 414) making up the structure of the structured result software object (34), by allocation of a new memory space or reutilisation of a free allocation, said creation being executed by a type manager agent (TM0, TM1, TM2) corresponding to the type of the element to be created.

16.Procedure according to one of the preceding claims, characterised in that the creation step for en element (410, 414) making up the structure of the structured result software object (34)

occurs between the read step for a data item (322, 327) indicating the creation of said element, and the first loading step for the same element.

17. Procedure according to one of the preceding claims, characterised in that the action corresponding to a data item (324, 325, 329), referred to as simple data item, namely not representing a tag, includes a storage step for the value of this data item in the memory space allocated to the current object.

18. Procedure according to one of the preceding claims, characterised in that, during the deserialisation procedure, an object index (4102) is assigned to at least one element (410) making up the structure of the structured result object (34), said object index identifying said element in a unique manner, so that said element can be designated or referenced by other objects or elements (414).

19. Procedure according to one of the preceding claims, characterised in that the action corresponding to a data item (3211) representing a tag, referred to as "REF" tag, indicating a reference, includes the following steps:

- read of at least one subsequent data item (3212) in the input stream (32);
- storage in the memory space (3441) allocated to the current object (344), following data already stored, of a data item designating an object (340) as source of the value of all or part of the current object, said source object or element (340) being identified by said subsequent data item (3212).

20.Procedure according to one of the preceding claims, characterised in that the loading of the current object is considered to be terminated when the data stored in the allocated memory space correspond to a given length, said length being read from storage in card (2) by a type manager agent (TM0, TM1, TM2) or by a generic type manager agent (TMG) in the embedded platform.

21.Procedure according to one of the preceding claims, characterised in that the transmission by the embedded platform (2) to the host (1) of a linear data sequence (42) of a given length, is executed according to an iterative procedure, referred to as data read procedure, which includes the following recursive steps:

- transmission of a communication command by the host, accompanied by at least one transmission parameter representing the length of the data already received;
- reception by the embedded platform of the transmission parameter, and comparison with the linear data sequence to be transmitted;
- response to the communication command by transmission of at least one return parameter (43), representing the data item or items following immediately after the data already received by the host;
- reception by the host of the communication command return parameter (43), and storage of the data which it represents after the data already received.

22. Procedure according to one of the preceding claims, characterised in that the reception by the embedded platform (2) from

the host (1) of a linear data sequence (43) of a given length, is executed according to an iterative procedure, referred to as data write procedure, comprising the following recursive steps:

- transmission by the host of a communication command accompanied by at least a first transmission parameter (32) representing the data item or items following immediately after the part of the linear data sequence to be transmitted already sent, and where appropriate, a second transmission parameter representing the position, in the sequence to be transmitted, of the data represented by the first transmission parameter, or the length of the data already transmitted;

- reception by the embedded platform of one or more communication command transmission parameters (32);

- storage in an input stream (33) in the embedded platform of the data represented by the first transmission parameter (32), after the data already received.

23. Procedure according to one of the preceding claims, characterised in that the data write procedure also includes the following steps:

- comparison by the embedded platform of the second transmission parameter, and comparison with the length of the data already received;

- return by the embedded platform of at least one response parameter representing either the length of the data already received, or a data item representing the result of this comparison, or both.

24.Procedure according to one of the preceding claims, characterised in that at least two of the serialisation, deserialisation, data read or data write procedures are executed in parallel, or interleaved by repetition of a step including successive execution of at least one step of each of said procedures.

25.Procedure according to one of the preceding claims, characterised in that the input stream or output stream, or both, are stored in the form of circular memory structures, it being possible for said two streams to share the same circular structure.

26. Procedure according to one of the preceding claims, characterised in that the embedded platform has a programmable environment, capable of storing and executing at least one application created by a programmer, the communication function being compatible with the APDU format defined in standard ISO 7816.

27. Conversion procedure according to the preceding claim, characterised in that deserialisation and treatment operations for an object received are initiated by reception of at least one APDU format command including at least one data item indicating reception of a structured object.

28. Procedure according to one of the preceding claims, characterised in that the embedded platform has a programmable environment compatible with the JavaCard® standard.

29. Procedure according to one of the preceding claims, characterised in that at least one of the applications executed on the host or embedded platform is programmed in Java® language.

30. Procedure according to one of the preceding claims, characterised in that the data read, data write, deserialisation or serialisation procedures are performed by implementation in at least one class stored in the host or embedded platform, said implementation including at least one of the following commands:

- an object write command, transmitting a structured object (31) to at least one agent (221) of the embedded platform, by utilisation of the serialisation procedure for this object, followed by the data write and deserialisation procedures;

- an object read command, reading a structured object (41) from at least one agent (221) of the embedded platform, by utilisation of the serialisation procedure, followed by the data read and deserialisation procedures;

- a deallocation command for a structured object stored in the embedded platform, by utilisation of the serialisation procedure, the latter also including a storage step for freeing or deallocation of the memory space allocated to each component of said object, after analysing the structure of this component;

- a housekeeping or erase command for a memory space freed by a structured object, by utilisation of the deserialisation procedure to create an object with non-material content from a given linear data sequence;

- a duplication command for a structured object, referred to as source object, by utilisation of the serialisation procedure without deallocation of said source object, to create a linear data sequence representing the same object, then by utilisation of the deserialisation procedure from said linear data sequence to create

another structured object the content of which is identical to that of the source object.

31. Procedure according to the preceding claim, characterised in that the programming language of the embedded platform includes a first class (IOApplet), describing a ProcessAPDU abstract method, initiating a user-definable process in the application on reception of an APDU message; the program code executing deserialisation operations in the embedded platform being stored in said embedded platform, as implementation of said ProcessABDU abstract method, in a second class (ObjectIOApplet) inheriting the first class (IOApplet), said program code calling a ProcessObject method, the latter being described as an abstract method in this same implementation class (ObjectIOApplet).

32. Procedure according to the claim (Javacard), characterised in that the programming language of the embedded platform includes a first class (IOApplet) describing a SendAPDU method transmitting an APDU format message to the host; the program code executing the serialisation operations in the embedded platform being stored, as an implementation of at least one SendObject method calling the SendAPDU method, in a second class (ObjectIOApplet) inheriting said first class (IOApplet).

33. Procedure according to one of the preceding claims, characterised in that it is used for communication between at least one software agent, referred to as card agent, stored or executed in the embedded platform, and at least one software agent, referred to as card engine proxy agent, stored or executed in at least one host belonging to a computer network communicating by asynchronous messages according to an AAA-MOM type software infrastructure,

said card engine proxy agent serving as an intermediary for the card agent in its communications with other agents of said network, said agents operating in accordance with the specifications of said software infrastructure and belonging to at least one distributed application.

34. Computer system comprising a computer station (2), referred to as embedded platform, comprising a portable object including at least a processor, storage facilities and communication resources capable of exchanging information with a terminal in the form of one or more linear data sequences, characterised in that the platform incorporates a serialisation agent (212), capable of executing a conversion step for a data set, in one direction or the other, between a linear data sequence arrangement on the one hand and a structured arrangement describing or representing one or more software objects structured or hierarchised in accordance with the criteria of an object-oriented programming language on the other hand.

35.System according to the previous claim, characterised in that the embedded platform (2) includes a communication agent (211) capable of:

- receiving, in place of the addressee agent (221), a data set (32) received by the response function (201) for an addressee software agent (221) stored in the embedded platform, said data set being arranged in one or more linear data sequences;
- converting said data set into at least one software object (34), structured or hierarchised according to the criteria of an object-oriented programming language;

- transmitting said structured software object (34) to the addressee agent (221), and initiating execution of a process (2210), according to said object, by said addressee agent (221).

36. System according to one of the preceding claims, characterised in that the linear data sequence representing a structured software object is stored in the embedded platform, in an input or output stream, the embedded platform incorporating a software agent, referred to as serialisation agent (212), capable of creating the structured object or objects represented by the input stream in the embedded platform, namely deserialising said structured objects, or writing data representing the structured object or objects to be transmitted in the output stream, namely serialising said structured objects.

37. System according to one of the preceding claims, characterised in that the input stream or output stream is stored in the form of one or more circular memory structures.

38. System according to one of the preceding claims, characterised in that the serialisation agent (212) uses a memory stack, referred to as type stack, to store the type of at least one object making up all or part of the structure of a structured object to be serialised or deserialised, said type stack including memory locations none of which are accessible until the locations loaded more recently have been read and erased.

39. System according to one of the preceding claims, characterised in that the data contained in the input or output stream represent one or more structured objects, using a code comprising a set of tags, each of said tags representing a given action to be executed on deserialisation of said linear data sequence.

40. System according to one of the preceding claims, characterised in that at least one tag is defined as representing one of the following actions:

- addition of a new element to the structure of the structured object represented by the linear data sequence;
- reference to an element or object, referred to as source object, as the source of the value of all or part of an element making up the structured object;
- indication that the following data item or items represent the content of an element making up the structured object;
- indication of the absence of content for an element making up the structured object.

41. System according to one of the preceding claims, characterised in that the embedded platform includes a portable object operating in accordance with standard ISO 7816 and using APDU format commands.

42. System according to one of the preceding claims, characterised in that at least one agent or application stored in the embedded platform is programmed in Java® language, the embedded platform having a computer environment in accordance with the JavaCard® standard.

43. System according to one of the preceding claims, characterised in that it includes at least one software class in the host, embedded platform or both, implementing at least one of the following commands:

- an object write command, transmitting a structured object (31) to at least one agent (221) of the card, by serialisation of said structured object into a data stream in the host, followed by transmission of this data stream to the embedded platform, and deserialisation of said data stream into a structured object in the embedded platform;

- an object read command, reading a structured object (41) from at least one agent (221) of the card, by serialisation of said structured object into a data stream in the embedded platform, followed by reception of said data stream from the embedded platform, and deserialisation of said data stream into a structured object in the host;

- a deallocation command for a structured object stored in the embedded platform, by a serialisation of said object according to an option involving freeing or deallocation of the memory space allocated to each component of said object, after analysis of the structure of said component;

- a housekeeping or erase command for a memory space freed by a structured object in the embedded platform, by deserialisation of a given linear data sequence to create an object with non-material content;

- a command for duplication in the embedded platform of a structured object, referred to as source object, by serialisation into a linear data sequence representing the same object, without deallocation of said source object, followed by deserialisation from said linear data sequence into another structured object the content of which is identical to that of the source object.

44. System according to one of the preceding claims, characterised in that the embedded platform communicates with at least one host belonging to a computer network communicating by asynchronous messages according to an AAA-MOM type software infrastructure, said host including a software agent, referred to as card engine proxy agent, capable of managing communications between said embedded platform and other agents of said network, said agents operating in accordance with the specifications of said software infrastructure and belonging to at least one distributed application.

Fig. 1

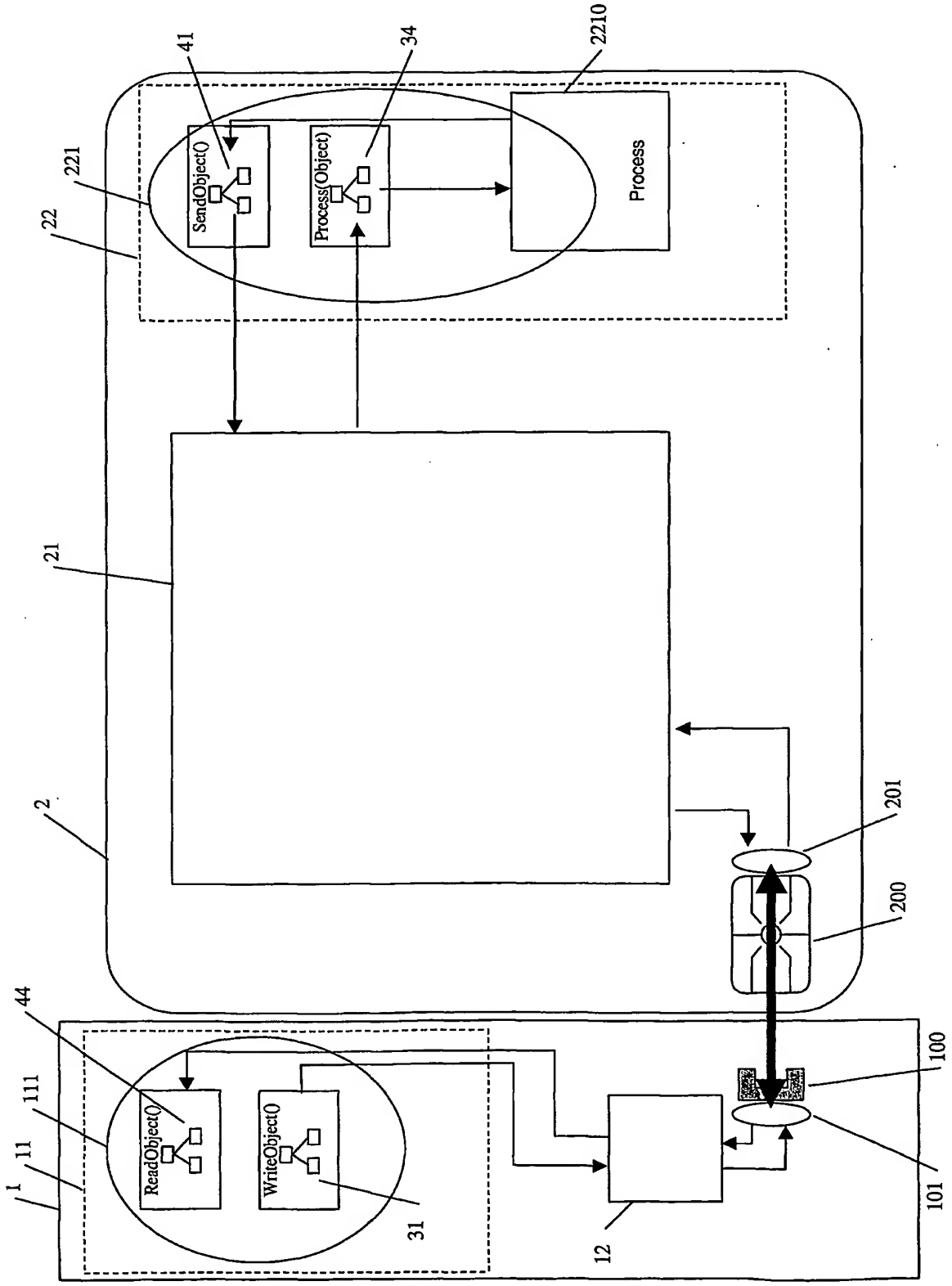

41
34
2210
221
22
SendObject()
Process(Object)
Process
21
2
201
200
44
111
11
1
ReadObject()
WriteObject()
31
12
100
101

Fig. 2

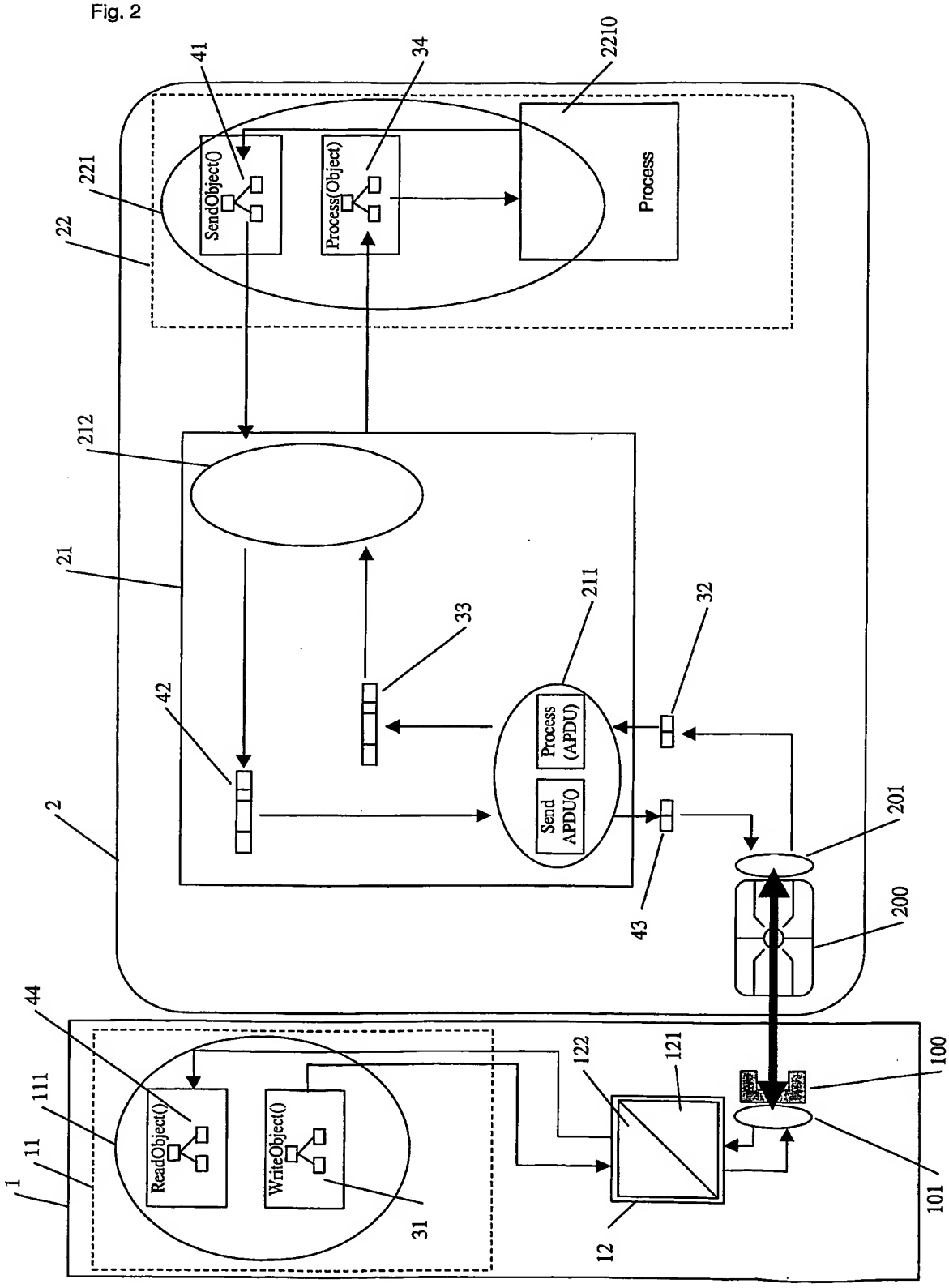

41
34
2210
221
22
SendObject()
Process(Object)
Process
212
21
211
33
32
42
Process (APDU)
Send APDU()
2
201
200
43
44
111
11
1
ReadObject()
WriteObject()
31
122
121
100
101
12

Fig. 3

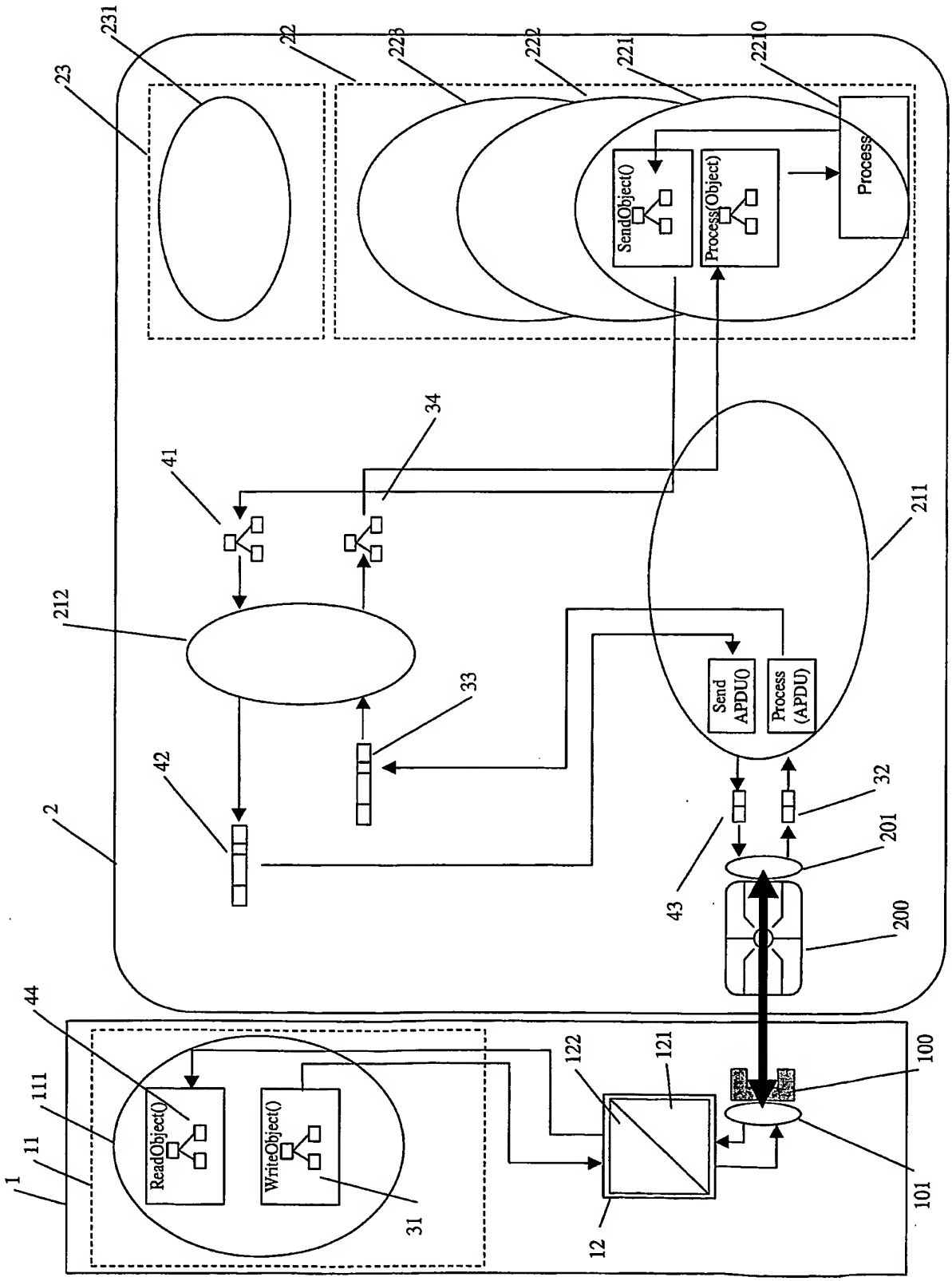

231
22
223
222
221
2210
23
SendObject()
Process(Object)
Process
34
41
211
212
Send APDU()
Process (APDU)
33
42
32
2
201
43
200
44
122
121
100
111
ReadObject()
WriteObject()
11
31
101
1
12

Fig. 4

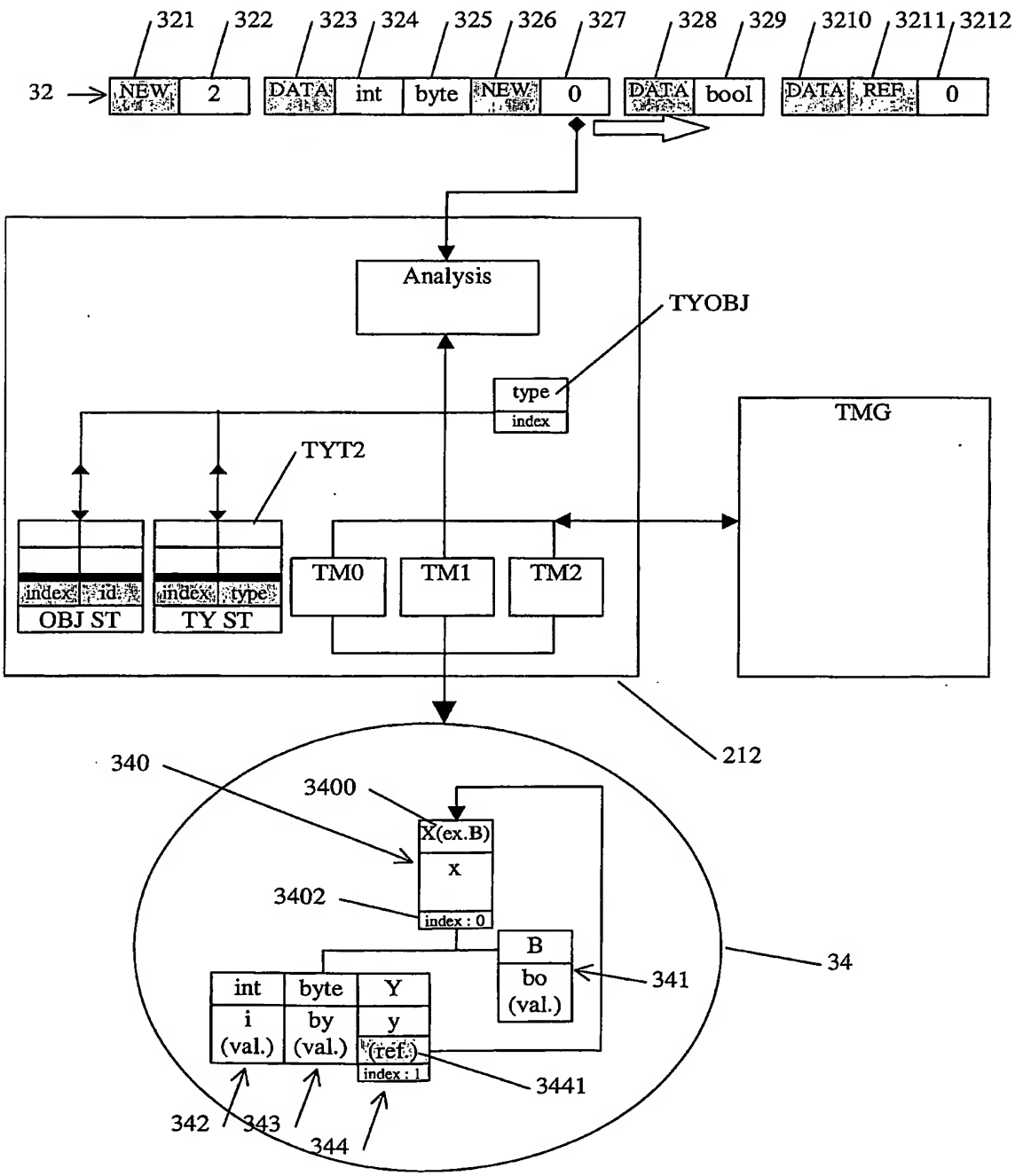
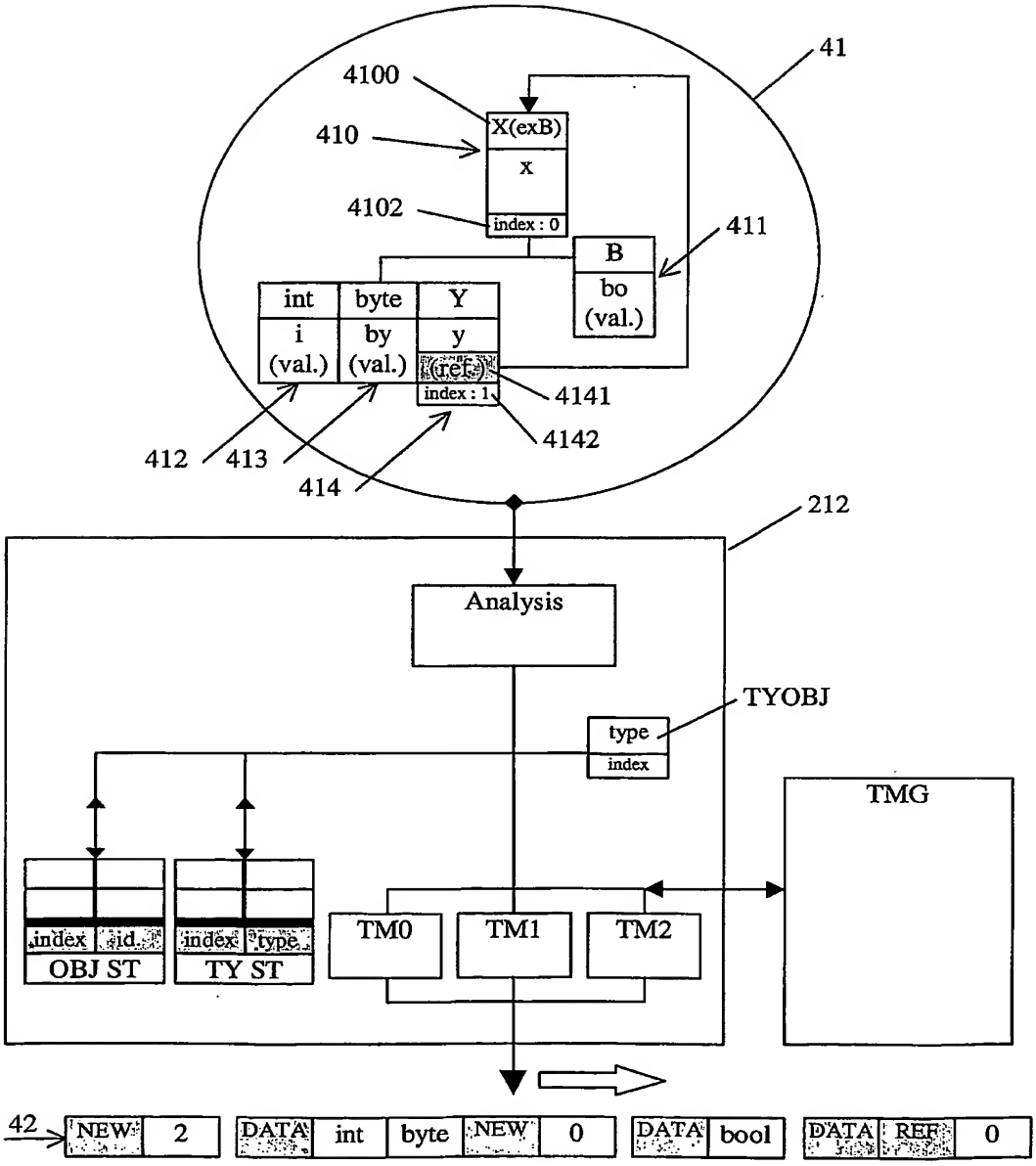

321
322
323
324
325
326
327
328
329
3210
3211
3212
32
NEW
2
DATA
int
byte
NEW
0
DATA
bool
DATA
REF
0
Analysis
TYOBJ
type
index
TMG
TYT2
index
id
OBJ ST
index
type
TY ST
TM0
TM1
TM2
212
340
3400
X(ex.B)
x
3402
index : 0
B
bo
(val.)
341
34
int
i
(val.)
byte
by
(val.)
Y
y
(ref.)
index : 1
3441
342
343
344

Fig. 5


41
4100
410
X(exB)
x
4102
index : 0
411
B
bo
(val.)
int
byte
Y
i
(val.)
by
(val.)
y
(ref.)
index : 1
4141
4142
412
413
414
212
Analysis
TYOBJ
type
index
TMG
index
id.
OBJ ST
index
type
TY ST
TM0
TM1
TM2
42
NEW
2
DATA
int
byte
NEW
0
DATA
bool
DATA
REF
0

INTERNATIONAL SEARCH REPORT

Internal'---' Application No
PCT/.. 03/00763

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G07F7/10 G06F9/44 G06F9/54

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G07F G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, INSPEC, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 00 68903 A (MICROSOFT CORP) 16 November 2000 (2000-11-16) abstract page 4, line 6 -page 5, line 8 page 13, line 13 -page 19, line 10; figures 2-4 claims 1-21 ---	1-44
A	DE 199 54 532 A (IBM) 8 June 2000 (2000-06-08) abstract page 3, line 43 -page 4, line 6 page 5, line 54 -page 8, line 54; figures 2,3 claims 1-11 --- -/--	1-44

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
31 October 2003	12/11/2003

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Wiltink, J

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Internat' ---' Application No
PCT/__ 03/00763

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	EP 1 071 219 A (IBM) 24 January 2001 (2001-01-24) abstract paragraphs '0005!-'0011!,'0016!,'0017! claims 1,4,8	1-44
A	URIEN P: "Internet card, a smart card as a true Internet node" COMPUTER COMMUNICATIONS, ELSEVIER SCIENCE PUBLISHERS BV, AMSTERDAM, NL, vol. 23, no. 17, 1 November 2000 (2000-11-01), pages 1655-1666, XP004238469 ISSN: 0140-3664 the whole document	1-44

INTERNATIONAL SEARCH REPORT

Internat'---' Application No
PCT/__ 03/00763

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 0068903 A	16-11-2000	US 6547150 B1	15-04-2003
		AU 4840400 A	21-11-2000
		WO 0068903 A1	16-11-2000
DE 19954532 A	08-06-2000	DE 19954532 A1	08-06-2000
EP 1071219 A	24-01-2001	DE 19933584 A1	18-01-2001
		EP 1071219 A2	24-01-2001

Copy for the designated Office (DO/EP) PCT/IB2003/000763

PATENT COOPERATION TREATY 31

From the INTERNATIONAL BUREAU

PCT

NOTIFICATION OF THE RECORDING OF A CHANGE

(PCT Rule 92bis.1 and Administrative Instructions, Section 422)

To:

AXALTO SA
c/o Patricia RENAULT
36-38 Rue de la Princesse
BP 45
F-78431 Louveciennes
France

Date of mailing (day/month/year)	IMPORTANT NOTIFICATION
15 September 2004 (15.09.2004)	
Applicant's or agent's file reference 71.03939 BC	
International application No. PCT/IB2003/000763	**International filing date** (day/month/year) 26 February 2003 (26.02.2003)

1. The following indications appeared on record concerning:

[X] the applicant [ ] the inventor [ ] the agent [ ] the common representative

Name and Address
SCHLUMBERGER SYSTEMES
50 Avenue Jean Jaures
F-92120 Montrouge
France

State of Nationality: FR
State of Residence: FR
Telephone No.: 33 1 30 08 47 81
Facsimile No.: 33 1 30 08 45 24
Teleprinter No.:

2. The International Bureau hereby notifies the applicant that the following change has been recorded concerning:

[ ] the person [X] the name [ ] the address [ ] the nationality [ ] the residence

Name and Address
AXALTO SA
50, avenue Jean Jaurès
F-92120 Montrouge
France

EPO -DG 1
22. 09. 2004
103

State of Nationality: FR
State of Residence: FR
Telephone No.: +33 1 46 00 48 98
Facsimile No.: +33 1 46 00 70 26
Teleprinter No.:

3. Further observations, if necessary:
CORRECTED VERSION OF THE NOTIFICATION FORM PCT/IB/306 MAILED ON 11 AUGUST 2004

4. A copy of this notification has been sent to:

[X] the receiving Office
[ ] the International Searching Authority
[ ] the International Preliminary Examining Authority
[X] the designated Offices concerned
[ ] the elected Offices concerned
[ ] other:

The International Bureau of WIPO
34, chemin des Colombettes
1211 Geneva 20, Switzerland
Facsimile No. (41-22) 338.70.60

Authorized officer
V. BLANC
Telephone No. (41-22) 338 9666

Copy for the designated Office (DO/EP) PCT/IB2003/000763

PATENT COOPERATION TREATY 31

From the INTERNATIONAL BUREAU

PCT

NOTIFICATION OF THE RECORDING OF A CHANGE

(PCT Rule 92bis.1 and Administrative Instructions, Section 422)

To:

AXALTO SA
c/o Patricia RENAULT
36-38 Rue de la Princesse
BP 45
F-78431 Louveciennes
France

Date of mailing (day/month/year) 11 August 2004 (11.08.2004)	
Applicant's or agent's file reference 71.03939 BC	**IMPORTANT NOTIFICATION**
International application No. PCT/IB2003/000763	**International filing date** (day/month/year) 26 February 2003 (26.02.2003)

1. The following indications appeared on record concerning:

[X] the applicant [ ] the inventor [ ] the agent [ ] the common representative

Name and Address	State of Nationality	State of Residence
SCHLUMBERGER SYSTEMES 50 Avenue Jean Jaures F-92120 Montrouge France	FR	FR
	Telephone No. 33 1 30 08 47 81	
	Facsimile No. 33 1 30 08 45 24	
	Teleprinter No.	

2. The International Bureau hereby notifies the applicant that the following change has been recorded concerning:

[X] the person [ ] the name [ ] the address [ ] the nationality [ ] the residence

Name and Address	State of Nationality	State of Residence
AXALTO SA 50, avenue Jean Jaurès F-92120 Montrouge France	FR	FR
	Telephone No. +33 1 46 00 48 98	
	Facsimile No. +33 1 46 00 70 26	
	Teleprinter No.	

EPO - DG 1
17. 08. 2004
103

3. Further observations, if necessary:
Please note that this change also applies to the common representative.

4. A copy of this notification has been sent to:

[X] the receiving Office
[ ] the International Searching Authority
[ ] the International Preliminary Examining Authority
[X] the designated Offices concerned
[ ] the elected Offices concerned
[ ] other:

The International Bureau of WIPO **34, chemin des Colombettes** **1211 Geneva 20, Switzerland**	Authorized officer V. BLANC
Facsimile No. (41-22) 338.70.60	Telephone No. (41-22) 338 9666

Form PCT/IB/306 (March 1994) 006389098

R E P U B L I Q U E F R A N Ç A I S E

PCT / IB 03 / 00763

10.03.03

INPI
INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE

BREVET D'INVENTION

CERTIFICAT D'UTILITÉ - CERTIFICAT D'ADDITION

COPIE OFFICIELLE

Le Directeur général de l'Institut national de la propriété industrielle certifie que le document ci-annexé est la copie certifiée conforme d'une demande de titre de propriété industrielle déposée à l'Institut.

Fait à Paris, le 03 MARS 2003

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)

Pour le Directeur général de l'Institut national de la propriété industrielle
Le Chef du Département des brevets

Martine PLANCHE

INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE
SIEGE
26 bis, rue de Saint Petersbourg
75800 PARIS cedex 08
Téléphone : 33 (0)1 53 04 53 04
Télécopie : 33 (0)1 53 04 45 23
www.inpi.fr
ETABLISSEMENT PUBLIC NATIONAL CREE PAR LA LOI N° 51-444 DU 19 AVRIL 1951

1er dépôt

INPI
INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE
26 bis, rue de Saint Pétersbourg
75800 Paris Cedex 08
Téléphone : 01 53 04 53 04 Télécopie : 01 42 94 86 54

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI

N° 11354*01

REQUÊTE EN DÉLIVRANCE 1/2

Cet imprimé est à remplir lisiblement à l'encre noire DB 540 W /260899

Réservé à l'INPI	
REMISE DES PIÈCES DATE 28 FEV 2002 LIEU 75 INPI PARIS N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI 0202570 DATE DE DÉPÔT ATTRIBUÉE PAR L'INPI 28 FEV. 2002	1 NOM ET ADRESSE DU DEMANDEUR OU DU MANDATAIRE À QUI LA CORRESPONDANCE DOIT ÊTRE ADRESSÉE CABINET DEBAY 126 ELYSEE 2 78170 LA CELLE SAINT CLOUD
Vos références pour ce dossier *(facultatif)* BULL 3939 FR	

Confirmation d'un dépôt par télécopie ☐ N° attribué par l'INPI à la télécopie

2 NATURE DE LA DEMANDE	Cochez l'une des 4 cases suivantes
Demande de brevet	☒
Demande de certificat d'utilité	☐
Demande divisionnaire	☐
Demande de brevet initiale	N° Date
ou demande de certificat d'utilité initiale	N° Date
Transformation d'une demande de brevet européen *Demande de brevet initiale*	☐ N° Date

3 TITRE DE L'INVENTION (200 caractères ou espaces maximum)

Procédé itératif de sérialisation d'objets logiciels structurés

4 DÉCLARATION DE PRIORITÉ OU REQUÊTE DU BÉNÉFICE DE LA DATE DE DÉPÔT D'UNE DEMANDE ANTÉRIEURE FRANÇAISE	
	Pays ou organisation Date N°
	Pays ou organisation Date N°
	Pays ou organisation Date N°
	☐ S'il y a d'autres priorités, cochez la case et utilisez l'imprimé «Suite»

5 DEMANDEUR		☒ S'il y a d'autres demandeurs, cochez la case et utilisez l'imprimé «Suite»	
Nom ou dénomination sociale		BULL S.A.	
Prénoms			
Forme juridique		Société Anonyme	
N° SIREN		6 4 2 0 5 8 7 3 9	
Code APE-NAF			
Adresse	Rue	68, route de Versailles	
	Code postal et ville	78430	LOUVECIENNES
Pays		FRANCE	
Nationalité		Française	
N° de téléphone *(facultatif)*			
N° de télécopie *(facultatif)*			
Adresse électronique *(facultatif)*			

1er dépôt

INPI
DIRECTION NATIONALE DE LA PROPRIÉTÉ INDUSTRIELLE

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

REQUÊTE EN DÉLIVRANCE 2/2

Réservé à l'INPI

REMISE DES PIÈCES DATE 28 FEV 2002
LIEU 75 INPI PARIS
N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI 0202570

DB 540 W /260899

Vos références pour ce dossier : *(facultatif)*		BULL 3939 FR
6 **MANDATAIRE**		
Nom		DEBAY
Prénom		Yves
Cabinet ou Société		CABINET DEBAY
N °de pouvoir permanent et/ou de lien contractuel		CPI 92-1066
Adresse	Rue	126 ELYSEE 2
	Code postal et ville	78170 LA CELLE SAINT CLOUD
N° de téléphone *(facultatif)*		01.39.18.46.24
N° de télécopie *(facultatif)*		01.39.18.67.08
Adresse électronique *(facultatif)*		Cab.Debay@wanadoo.fr
7 **INVENTEUR (S)**		
Les inventeurs sont les demandeurs		☐ Oui ☒ Non **Dans ce cas fournir une désignation d'inventeur(s) séparée**
8 **RAPPORT DE RECHERCHE**		**Uniquement pour une demande de brevet (y compris division et transformation)**
Établissement immédiat ou établissement différé		☒ ☐
Paiement échelonné de la redevance		**Paiement en trois versements, uniquement pour les personnes physiques** ☐ Oui ☐ Non
9 **RÉDUCTION DU TAUX DES REDEVANCES**		**Uniquement pour les personnes physiques** ☐ Requise pour la première fois pour cette invention *(joindre un avis de non-imposition)* ☐ Requise antérieurement à ce dépôt *(joindre une copie de la décision d'admission pour cette invention ou indiquer sa référence)* :
Si vous avez utilisé l'imprimé «Suite», indiquez le nombre de pages jointes		1

10 **SIGNATURE DU DEMANDEUR OU DU MANDATAIRE (Nom et qualité du signataire)**
Y. DEBAY Mandataire (CPI 92-1066)

VISA DE LA PRÉFECTURE OU DE L'INPI

La loi n°78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés s'applique aux réponses faites à ce formulaire.
Elle garantit un droit d'accès et de rectification pour les données vous concernant auprès de l'INPI.

1er dépôt

INPI
INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE
26 bis, rue de Saint Pétersbourg
75800 Paris Cedex 08
Téléphone : 01 53 04 53 04 Télécopie : 01 42 94 86 54

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI

N° 11354*01

REQUÊTE EN DÉLIVRANCE
Page suite N° 1../1..

Réservé à l'INPI
REMISE DES PIÈCES
DATE 28 FEV 2002
LIEU 75 INPI PARIS
N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI 0202570

Cet imprimé est à remplir lisiblement à l'encre noire

DB 829 W /260399

Vos références pour ce dossier *(facultatif)*		BULL 3939 FR
4 DÉCLARATION DE PRIORITÉ OU REQUÊTE DU BÉNÉFICE DE LA DATE DE DÉPÔT D'UNE DEMANDE ANTÉRIEURE FRANÇAISE		Pays ou organisation Date __/__/____ N° Pays ou organisation Date __/__/____ N° Pays ou organisation Date __/__/____ N°
5 DEMANDEUR		
Nom ou dénomination sociale		INRIA
Prénoms		
Forme juridique		Etablissement public
N° SIREN		| . . . . . |
Code APE-NAF		| . . . . |
Adresse	Rue	Domaine de Voluceau Rocquencourt B.P. 105
	Code postal et ville	78153 LE CHESNAY CEDEX
Pays		FRANCE
Nationalité		Française
N° de téléphone *(facultatif)*		
N° de télécopie *(facultatif)*		
Adresse électronique *(facultatif)*		
5 DEMANDEUR		
Nom ou dénomination sociale		
Prénoms		
Forme juridique		
N° SIREN		| . . . . . |
Code APE-NAF		| . . . . |
Adresse	Rue	
	Code postal et ville	
Pays		
Nationalité		
N° de téléphone *(facultatif)*		
N° de télécopie *(facultatif)*		
Adresse électronique *(facultatif)*		

10 SIGNATURE DU DEMANDEUR OU DU MANDATAIRE (Nom et qualité du signataire)
Yves DEBAY (CPI 92-1066)

VISA DE LA PRÉFECTURE OU DE L'INPI

La loi n°78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés s'applique aux réponses faites à ce formulaire.
Elle garantit un droit d'accès et de rectification pour les données vous concernant auprès de l'INPI

Procédé itératif de sérialisation d'objets logiciels structurés

La présente invention concerne un procédé itératif de conversion d'objets logiciels structuré en un flux de données brutes et inversement, permettant leur transfert direct par des moyens de communication simples comme ceux d'une station informatique embarquée, ainsi qu'une remise à zéro de tels objets logiciels ou une réutilisation de l'espace mémoire qui leur est alloué.

Au sein d'une application programmée dans un langage de haut niveau, et en particulier dans le cas des langages structurés ou orientés objets comme par exemple Java®, une grande part des données utilisées ou traitées sont organisées et mémorisées sous la forme d'objets logiciels comportant une structure précise, plus complexe qu'une simple succession d'octets. De tels objets peuvent regrouper plusieurs variables ou groupes de données, par exemple sous la forme de types simples, caractères (type "char"), entier (type "int"), ou de types d'objets de base ou définis par le programmeur, chaîne de caractères (type "String"), ou de tableaux (type "array") à une ou plusieurs dimensions des types précédemment définis. De tels objets structurés peuvent donc être définis ou représentés par une arborescence regroupant différents objets eux-mêmes de différents types, l'organisation d'une telle structure étant parfois appelée le graphe de composition de l'objet. Un type d'objet peut être défini "par héritage" à partir de la définition d'un autre type (généralement appelé "super-type"), l'ensemble des types peut être représenté sous la forme d'une arborescence nommée le graphe d'héritage.

Selon les cas, il peut être utile de pouvoir transférer, créer, ou supprimer les objets logiciels maniés par une application informatique, dans des environnements matériels ou logiciels où ces fonctionnalités de sont pas toujours possibles ou pratiques.

De nombreuses applications informatiques, par exemple, comportent une communication entre d'une part un terminal informatique et éventuellement un réseau informatique, et d'autre part un objet portable ou plate-forme embarquée ayant des capacités de traitement de données, comme par exemple une carte à puce de gestion bancaire, d'abonnement téléphonique ou de santé, ou un dispositif de repérage, de marquage, ou de péage informatisé.

De tels objets portables comportent en particulier un processeur associé à des moyens de mémorisation, contenant au moins un programme d'application, et à des moyens de communication, avec un ou plusieurs terminaux. Ces moyens de communication sont basés par exemple sur une transmission d'informations électroniques par différents types de technologies, telles que contact électrique, antenne radio, transmission lumineuse ou autre, plusieurs types de transmissions pouvant être combinées dans le même objet portable. Pour des raisons d'encombrement ou parfois d'ordre historique, les moyens de communications utilisés de façon courante fonctionnent selon des protocoles simples, par exemple « APDU » selon la norme ISO 7816 pour les cartes à puces. Ces protocoles ne permettent parfois le transfert que d'objets de types simples (sous forme d'entiers, voire de caractères simples) en tant que paramètres de commande transmis à l'objet ou obtenus en réponse de sa part, ou seulement à l'initiative du terminal, ou les deux. Dans le cas de la norme ISO 7816, le protocole APDU ne permet que le transfert d'objets sans types en tant que données brutes, sous forme d'octets transmis à l'objet portable informatisé en tant que paramètres de commande, ou obtenus en réponse de sa part, et seulement à l'initiative du terminal.

En l'état actuel de leur évolution, ces objets portables de traitement de données sont pour certains dotés d'un fonctionnement interne, par exemple JavaCard®, apte à exploiter directement des éléments d'applications programmés dans des langages de haut niveau voire orientés objet, par exemple Java®, et à être intégrés dans des applications centralisées ou distribuées communiquant de façon étendue. Pour communiquer avec des applications situées à l'intérieur d'une telle plate-forme embarquée, le programmeur d'une telle application doit alors s'abstenir d'utiliser des objets

logiciels structurés, ou prévoir de traiter directement et au cas par cas le transfert de ces objets avec cette plate-forme embarquée.

Pour bénéficier des performances et de la souplesse d'un tel langage lors de la programmation d'une application exécutable dans le processeur d'un tel objet portable, il devient donc utile que cette application puisse communiquer facilement avec d'autres applications externes à cet objet. Il est donc utile pour cela qu'elle puisse échanger avec elles les objets qu'elles traitent, sans perte de l'organisation et de la structure de ces objets.

Dans des langages de programmation tels que Java (marque déposée), les procédés utilisés pour la sérialisation ou désérialisation d'objets structurés peuvent compter sur des ressources matérielles et logicielles importantes. Ces procédés sont utilisés en particulier pour sauvegarder un objet structuré dans un fichier, ou pour transmettre un objet structuré entre deux processus de programme s'exécutant dans deux zones mémoires de travail différentes.

Ces procédés utilisent toutefois des ressources logicielles et matérielles qui ne sont pas disponibles dans certaines plates-formes embarquées, en particulier Javacard (marque déposée). A titre d'exemple, les classes de bases utilisées par une plate-forme Java classique occupent une taille mémoire supérieure au méga-octet (Java Developpement Kit : 9 Mo), alors qu'une carte à puce au standard Javacard peut ne disposer que de 16 kilo-octets.

Les procédés classiques de sérialisation utilisent les ressources classiques de l'environnement Java et leurs algorithmes sont conçus avec des objectifs de facilité d'emploi et de maintenance informatique. Ils sont donc beaucoup trop gourmands en mémoire et beaucoup trop complexes pour pouvoir être transposés dans une plate-forme embarquée d'aussi faible performance en termes de capacité mémoire et de puissance et vitesse de processeur.

Par ailleurs, en Java par exemple, les procédés de sérialisation utilisent un gestionnaire générique de types d'objets (« type manager »), implémenté au

sein de la machine virtuelle (VM) Java s'exécutant sur chaque plate-forme matérielle.

Du fait de la concision nécessaire pour être adaptée à une plate-forme embarquée, l'environnement Javacard dans ses versions actuelles ne comporte pas de gestionnaire de types. Un procédé de sérialisation tel qu'utilisé en Java standard ne disposerait donc pas des informations représentant la structure des objets structurés à reconstituer à partir des données reçues en format APDU.

Par ailleurs, dans un environnement informatique classique, les ressources en mémoire sont telles que les procédés de sérialisation implémentés dans les langages de programmation orientés objets, comme Java, ne se préoccupent pas de la taille des objets structurés à transmettre. Les objets à envoyer sont sérialisés par l'émetteur indépendamment de leur réception par le destinataire, et les objets reçus peuvent être désérialisés par le destinataire indépendamment de l'émetteur ou du moment de leur transmission. Les flux linéaires de données sont alors transmis et gérés entre émetteur et destinataire par des mécanismes logiciels qui sont extérieurs à l'environnement de programmation. Ces flux peuvent ainsi transiter par des zones tampons en mémoire (« buffers ») de capacités importantes, et sont gérés par d'autres « couches » logicielles, par exemple par un protocole tel que TCP/IP.

Dans le cas d'une plate-forme embarquée, les mécanismes logiciels permettant d'échanger des données avec l'extérieur n'offrent pas de fonctionnalités de gestion de flux assurant l'intégrité et la continuité des données transmises. Les faibles ressources en mémoire d'une plate-forme embarquée ne permettent pas non plus de mémoriser des flux linéaires donc des objets de taille importante au cours de la transmission et des conversions.

Un des buts de la présente invention est donc de proposer un procédé permettant au programmeur d'une application utilisant une plate-forme embarquée de disposer d'outils logiciels automatisés permettant à un agent logiciel, ou élément d'application, mémorisé ou exécuté dans un tel objet

portable, de recevoir ou d'envoyer à un autre agent, situé sur une autre station informatique, des données organisées sous la forme d'objets logiciels structurés, lorsque les moyens de communications entre eux ne permettent pas le transfert en tant que telles des structures qui les composent, mais seulement le transfert de données sous une forme plus simple, et que les ressources logicielles et matérielles de cette plate-forme embarquée ne sont pas suffisantes pour permettre l'utilisation d'un procédé classique de sérialisation.

Cet objectif est atteint par un procédé de conversion de données utilisable par une station informatique, dite plate-forme embarquée, comprenant un objet portable incluant au moins un processeur, des moyens de mémorisation, et des moyens de communication aptes à échanger des informations avec un terminal sous la forme d'une ou plusieurs successions linéaires de données, caractérisé en ce qu'il comporte une étape de conversion d'un ensemble de données, dans un sens ou dans l'autre, entre d'une part un agencement en une succession linéaire de données et d'autre part un agencement structuré décrivant ou représentant un ou plusieurs objets logiciels structurés ou hiérarchisés suivant les critères d'un langage de programmation orienté objet.

Selon une particularité, le procédé comporte des étapes de :

- conversion, ou sérialisation, d'un premier ensemble de données à transmettre comportant ou représentant un ou plusieurs objets logiciels structurés ou hiérarchisés suivant les critères d'un langage de programmation orienté objet, depuis un agencement structuré décrivant ou représentant cet objet vers une succession linéaire de données représentant ce premier ensemble de données ;
- transmission de cette succession linéaire de données par des moyens de communication, depuis la plate-forme embarquée vers au moins un hôte, c'est à dire un terminal ou une station informatique reliée au terminal, ou de cet hôte vers la plate-forme embarquée ;
- conversion après transmission, ou dé-sérialisation, de cette succession linéaire de données vers un ensemble de données agencé en un ou

plusieurs objets logiciels structurés reproduisant ou représentant le premier ensemble de données.

Selon une particularité, le terminal hôte envoie des informations à la plate-forme embarquée en utilisant un agent logiciel, dit fonction de transmission, ces informations étant reçues dans la plate-forme embarquée par une fonction de réponse apte à déclencher un traitement de ces données par au moins un agent logiciel destinataire mémorisé dans la plate-forme embarquée et faisant partie d'au moins une application, le procédé comprenant des étapes de :

- réception, par un agent de communication en lieu et place de l'agent destinataire, d'un ensemble de données reçues par la fonction de réponse à l'intention de cet agent logiciel destinataire, cet ensemble de données étant agencé en une succession linéaire de données ;
- conversion de cet ensemble de données en au moins un objet logiciel, structuré ou hiérarchisé suivant les critères d'un langage de programmation orienté objet ;
- transmission de cet objet logiciel structuré à l'agent destinataire et déclenchement d'un traitement en fonction de cet objet par cet agent destinataire.

Selon une particularité, le procédé comporte des étapes de :

- réception par la fonction de réponse, en provenance de la fonction de transmission de l'hôte, d'au moins une donnée sous forme d'au moins un paramètre d'envoi, et transmission de ce paramètre à un agent de communication mémorisé ou exécuté dans la plate-forme embarquée ;
- conversion, ou concaténation, par l'agent de communication d'au moins un paramètre d'envoi, transmis par l'agent de réponse, en un ensemble de données agencé en une succession linéaire de données et mémorisation de ces données dans un flux d'entrée dans la plate-forme embarquée ;
- conversion, ou dé-sérialisation, par un agent de sérialisation, mémorisé ou exécuté dans la plate-forme embarquée, d'au moins une partie des données mémorisées dans ce flux d'entrée vers un ensemble de données comprenant ou représentant au moins un objet logiciel structuré ;

- réception de cet objet logiciel structuré ou de ses références par l'agent destinataire.

Selon une particularité, le procédé comporte des étapes de :

- transmission d'un objet logiciel structuré ou de sa représentation depuis un agent logiciel faisant partie d'une application exécutée ou mémorisée dans une plate-forme embarquée vers un agent de sérialisation exécuté ou mémorisé dans cette plate-forme embarquée ;
- conversion, ou sérialisation, par cet agent de sérialisation de cet objet logiciel structuré vers un ensemble de données agencé en une succession linéaire de données et mémorisation de ces données dans un flux de sortie dans la plate-forme embarquée ;
- conversion par un agent de communication, mémorisé ou exécuté dans la plate-forme embarquée, d'au moins une partie des données mémorisées dans ce flux de sortie vers un ensemble de paramètres de réponse aptes à être transmis par la fonction de réponse ;
- transmission de ces paramètres de réponse depuis la plate-forme embarquée vers le terminal hôte par la fonction de réponse, de sa propre initiative ou en réponse à la fonction de transmission du terminal hôte.

Selon une particularité, la succession linéaire de données mémorisée dans le flux d'entrée ou le flux de sortie représente un ou plusieurs objets logiciels structurés ou hiérarchisés en utilisant une ou plusieurs données, dites balises, ayant une ou plusieurs valeurs déterminées représentant chacune une action déterminée à effectuer lors de la désérialisation de cette succession linéaire de données.

Selon une particularité, au moins une balise est définie comme représentant une des actions suivantes :

- ajout d'un nouvel élément à la structure de l'objet structuré représenté par la succession linéaire de données ;
- référence à un élément ou objet, dit objet source, en tant que source de la valeur de tout ou partie d'un élément composant l'objet structuré ;
- indication que la ou les données suivantes représentent le contenu d'un élément composant l'objet structuré ;

- indication d'une absence de contenu d'un élément composant l'objet structuré.

Selon une particularité, l'agent de sérialisation effectue la sérialisation d'un objet structuré, dit objet de départ, en un ensemble linéaire de données suivant un procédé, dit procédé de sérialisation, traitant au moins un des objets, dits éléments, composant la structure ou l'arborescence de cet objet structuré de départ, selon des étapes de :

- détection, par l'agent de sérialisation du type d'un élément, dit objet en cours, composant la structure ou l'arborescence de cet objet structuré ;
- mémorisation dans le flux de sortie d'une donnée représentant une balise indiquant l'ajout d'un nouvel élément, suivie d'une donnée représentant le type de l'objet en cours ;
- mémorisation dans le flux de sortie, à la suite des éléments qui y sont déjà présents, par un agent de sérialisation de type associé au type de l'objet en cours,
  - soit d'au moins une donnée représentant la valeur de tout ou partie de l'objet structuré ;
  - soit d'au moins une donnée représentant une balise indiquant une référence à un objet en tant que source de la valeur de tout ou partie de l'objet structuré, cette balise étant suivie d'une donnée identifiant ledit objet source ;

Selon une particularité, le procédé de sérialisation effectue la conversion d'un objet structuré vers le flux de sortie en mémorisant au fur et à mesure de ses itérations le type de chaque objet en cours dans une pile mémoire, dite pile de types, dont les emplacements sont lus dans l'ordre inverse de leur ordre de mémorisation.

Selon une particularité, l'agent de sérialisation effectue la désérialisation d'un ensemble linéaire de données en au moins un objet structuré résultat, suivant un procédé, dit procédé de désérialisation, traitant chacune des données mémorisées dans le flux d'entrée, selon des étapes de :

- lecture par l'agent de sérialisation d'au moins une donnée mémorisée dans le flux d'entrée à la suite des données précédemment traitées ;

- analyse de cette donnée et réalisation d'une action correspondant à cette donnée.

Un des buts de l'invention est également de proposer un tel procédé de transfert d'objets structuré permettant le transfert d'objets logiciels entre un hôte et une plate-forme embarquée lorsque l'environnement de fonctionnement de cette plate-forme embarquée ne comporte pas d'agent gestionnaire de types pour de tels objets structurés.

Ce but est atteint par le procédé de conversion de données décrit ci-dessus, caractérisé en ce que le procédé de désérialisation comprend le remplissage d'un élément, dit objet en cours, c'est à dire l'affectation d'une valeur directe ou indirecte à tout ou partie de cet objet en cours, cet élément composant tout ou partie de la structure de l'objet structuré résultat, la fin du remplissage de l'objet en cours déclenchant

- la lecture d'une donnée représentant un type d'objet mémorisé dans le prochain emplacement d'une structure de mémoire, dite pile de types, et l'effacement de cette donnée de cet emplacement ;
- la mémorisation, comme type d'un nouvel objet en cours, du type représenté par la donnée lue dans la pile de types.

Selon une particularité, l'objet structuré créé par le procédé de désérialisation est considéré comme complet et transmis à l'agent logiciel ou l'application qui en est destinataire lorsque la pile de types est vide.

Selon une particularité, l'action de désérialisation correspondant à une donnée représentant une balise, dite balise « NEW », indiquant un nouvel élément comprend des étapes de :

- lecture d'au moins une donnée suivante dans le flux d'entrée ;
- mémorisation du type d'objet, représenté par cette même donnée suivante, dans une pile mémoire, dite pile de types, à la suite des types qui y sont déjà éventuellement mémorisés ;

Selon une particularité, la pile de types utilisée par le procédé de désérialisation comprend une pile de type LIFO, c'est à dire dont les emplacements sont lus dans l'ordre inverse de leur ordre de mémorisation.

Selon une particularité, le procédé de désérialisation comprend une étape de création d'un élément composant la structure de l'objet logiciel structuré résultat, par allocation d'un nouvel espace mémoire ou par réutilisation d'une allocation libre, cette création étant réalisée par un agent de contrôle de type correspondant au type de l'élément à créer.

Selon une particularité, l'étape de création d'un élément composant la structure de l'objet logiciel structuré résultat a lieu entre l'étape de lecture d'une donnée indiquant la création de cet élément et la première étape de remplissage de ce même élément.

Selon une particularité, l'action correspondant à une donnée, dite donnée simple, c'est à dire ne représentant pas une balise, comprend une étape de mémorisation de la valeur de cette donnée dans l'espace mémoire alloué à l'objet en cours.

Selon une particularité, au cours du procédé de désérialisation, un index d'objet est attribué à au moins un élément composant la structure de l'objet structuré résultat, cet index d'objet identifiant cet élément de façon unique et lui permettant d'être désigné ou référencé par d'autres objets ou éléments.

Selon une particularité, l'action correspondant à une donnée représentant une balise, dite balise « REF », indiquant une référence comprend des étapes de :

- lecture d'au moins une donnée suivante dans le flux d'entrée ;
- mémorisation dans l'espace mémoire alloué à l'objet en cours, à la suite des données déjà mémorisées, d'une donnée désignant un objet comme source de la valeur de tout ou partie de l'objet en cours, cet objet ou élément source étant identifié par ladite donnée suivante.

Selon une particularité, le remplissage de l'objet en cours est considéré comme terminé lorsque les données mémorisées dans l'espace mémoire qui lui est alloué correspondent à une longueur déterminée, cette longueur étant lue en mémoire dans la carte par un agent de contrôle de type ou par un agent gestionnaire de types mémorisé dans la plate-forme embarquée.

Selon une particularité, l'envoi par la plate-forme embarquée, vers l'hôte, d'une succession linéaire de données d'une longueur déterminée, s'effectue selon un procédé itératif, dit de lecture de données, comprenant des étapes récursives de :

- envoi par l'hôte d'une commande de communication avec passage d'au moins un paramètre d'envoi représentant la longueur des données déjà reçues ;
- réception par la plate-forme embarquée du paramètre d'envoi et comparaison avec la succession linéaire de données à transmettre ;
- réponse à la commande de communication par l'envoi d'au moins un paramètre de retour représentant la ou les données suivant immédiatement les données déjà reçues par l'hôte ;
- réception par l'hôte du paramètre de retour de la commande de communication et mémorisation des données qu'il représente à la suite des données déjà reçues.

Selon une particularité, la réception par la plate-forme embarquée, en provenance de l'hôte, d'une succession linéaire de données d'une longueur déterminée, s'effectue selon un procédé itératif, dit d'écriture de données, comprenant des étapes récursives de :

- envoi par l'hôte d'une commande de communication avec passage d'au moins un premier paramètre d'envoi représentant la ou les données suivant immédiatement la partie déjà transmise de la succession linéaire de données à transmettre, ainsi éventuellement qu'un deuxième paramètre d'envoi représentant la position, dans la succession à transmettre, des données représentées par le premier paramètre d'envoi ou la longueur des données déjà transmises ;
- réception par la plate-forme embarquée du ou des paramètres d'envoi de la commande de communication ;
- mémorisation dans un flux d'entrée dans la plate-forme embarquée des données représentées par le premier paramètre d'envoi à la suite des données déjà reçues.

Selon une particularité, le procédé d'écriture de données comprend en outre les étapes suivantes :

- comparaison par la plate-forme embarquée du deuxième paramètre d'envoi et comparaison avec la longueur des données déjà reçues ;
- retour par la plate-forme embarquée d'au moins un paramètre de réponse représentant soit la longueur des données déjà reçues soit une donnée représentant le résultat de cette comparaison soit les deux ;

Un des buts de l'invention est également de proposer un tel procédé de transfert d'objets structuré permettant le transfert d'objets logiciels d'une taille importante par rapport aux possibilités des moyens de communication en matière de transfert ou de stockage provisoire, ou d'objets logiciels de taille illimitée.

Ce but est atteint par le procédé de conversion de données décrit ci-dessus, caractérisé en ce qu'au moins deux des procédés de sérialisation, désérialisation, lecture de données, ou écriture de données sont effectués en parallèle ou de façon entrelacée par répétition d'une étape comprenant l'exécution successive d'au moins une étape de chacun de ces procédés.

Selon une particularité, le flux d'entrée ou le flux de sortie ou les deux sont mémorisés sous la forme de structures circulaires de mémoire, ces deux flux pouvant partager la même structure circulaire.

Selon une particularité, la plate-forme embarquée comporte un environnement programmable apte à mémoriser et exécuter au moins une application réalisée par un programmeur, la fonction de communication étant compatible avec le format « APDU » défini dans la norme ISO 7816.

Selon une particularité, des opérations de désérialisation puis traitement d'un objet reçu sont déclenchées par la réception d'au moins une commande au format APDU comprenant au moins une donnée indiquant la réception d'un objet structuré.

Selon une particularité, la plate-forme embarquée comporte un environnement programmable compatible avec le standard JavaCard (marque déposée).

Selon une particularité, au moins une des applications exécutées sur l'hôte ou la plate-forme embarquée est programmée en langage Java®.

Selon une particularité, le procédé est utilisé pour la communication entre au moins un agent logiciel, dit agent de carte, mémorisé ou exécuté dans la plate-forme embarquée et au moins un agent logiciel, dit agent proxy de moteur de carte mémorisé ou exécuté dans au moins un hôte appartenant à un réseau informatique communiquant par messages asynchrones selon une infrastructure logicielle de type AAA-MOM, cet agent proxy de moteur de carte servant d'intermédiaire à l'agent de carte dans ses communications avec d'autres agents de ce réseau, ces agents fonctionnant selon les spécifications de cette infrastructure logicielle et appartenant à au moins une application distribuée.

Par ailleurs, dans certains environnements ou systèmes d'exploitation, par exemple de type embarqué comme JavaCard®, il n'existe pas de procédure, ou seulement des procédures complexes et coûteuses en termes de ressources, pour supprimer un tel objet logiciel une fois qu'il n'est plus utilisé ou libérer l'espace qu'il occupe dans les moyens de mémorisation. Cette suppression doit alors être faite « manuellement », c'est à dire être prévue directement et au cas par cas par le programmeur de l'application.

Un des buts de la présente invention est donc de proposer un procédé permettant au programmeur d'une application utilisant une plate-forme embarquée de disposer d'outils logiciels permettant la réutilisation de l'espace mémoire occupé par tout ou partie de certains objets logiciels structurés utilisés par une application dans une station de traitement de données portable ou non.

Il peut également être utile de disposer d'outils logiciel permettant de réaliser aisément la duplication d'un objet logiciel comportant une structure non linéaire, par exemple sous forme d'arborescence.

Un des buts de la présente invention est donc de proposer un procédé permettant au programmeur d'une application utilisant une plate-forme embarquée de disposer d'outils logiciels permettant la duplication d'un objet logiciel structuré, dit de départ, vers un autre objet, dit résultat, contenant les

mêmes valeurs que l'objet de départ mais constituant un objet différent en mémoire.

De même, pour des raisons de sécurité, lorsque l'espace occupé par un tel objet logiciel se libère, il peut être important d'effacer de cet espace les informations provenant de cet objet, pour éviter que ces informations puissent être lues par un autre objet ou une autre application réutilisant ce même espace plus tard. Cet effacement doit alors être fait « manuellement », c'est à dire être prévu directement et au cas par cas par le programmeur de l'application.

Un des buts de la présente invention est donc de proposer un procédé permettant au programmeur d'une application utilisant une plate-forme embarquée de disposer d'outils logiciels permettant l'effacement ou l'uniformisation des informations contenues dans l'espace mémoire utilisé par un tel objet logiciel lors de la suppression de celui-ci ou de la réutilisation de cet espace.

L'invention propose alors un procédé comme décrit plus haut, caractérisé en ce que les procédés de lecture de données, d'écriture de données, de dé-sérialisation ou de sérialisation sont mis en œuvre à travers leur implémentation dans au moins une classe mémorisée dans l'hôte ou dans la plate-forme embarquée, cette implémentation comprenant au moins l'une des commandes suivantes :

- une commande d'écriture d'objet, effectuant la transmission d'un objet structuré à au moins un agent de la plate-forme embarquée, par utilisation du procédé de sérialisation de cet objet, puis du procédé d'écriture de données, puis du procédé de dé-sérialisation ;
- une commande de lecture d'objet, effectuant la lecture d'un objet structuré depuis au moins un agent de la plate-forme embarquée, par utilisation du procédé de sérialisation puis du procédé de lecture de données, puis du procédé de désérialisation ;
- une commande de désallocation d'un objet structuré mémorisé dans la plate-forme embarquée, par utilisation du procédé de sérialisation, celui-ci comprenant en outre une étape de mémorisation de la libération ou

désallocation de l'espace mémoire alloué à chaque composant de cet objet, après analyse de la structure de ce composant ;

- une commande de nettoyage ou effacement d'un espace mémoire libéré par un objet structuré, par utilisation du procédé de désérialisation pour créer un objet de contenu sans signification à partir d'une succession linéaire de données déterminée ;
- une commande de duplication d'un objet structuré dit de départ, par utilisation du procédé de sérialisation, sans désallocation de cet objet de départ, pour créer une succession linéaire de données représentant ce même objet, puis par utilisation du procédé de désérialisation à partir de cette succession linéaire de données pour créer un autre objet structuré de contenu identique à l'objet de départ.

Selon une particularité, le langage de programmation de la plate-forme embarquée comporte une première classe (IOApplet) décrivant une méthode abstraite ProcessAPDU lançant, lors de la réception d'un message APDU, un traitement paramétrable dans l'application ; le code programme réalisant les opérations de désérialisation dans la plate-forme embarquée étant mémorisé dans la plate-forme embarquée, en tant qu'implémentation de cette méthode abstraite ProcessAPDU, dans une deuxième classe (ObjectIOApplet) héritant de la première classe (IOApplet), ce même code programme faisant appel à une méthode ProcessObject elle-même décrite comme une méthode abstraite dans cette même classe (ObjectIOApplet) d'implémentation.

Selon une particularité, le langage de programmation de la plate-forme embarquée comporte une première classe (IOApplet) décrivant une méthode SendAPDU émettant vers l'hôte un message au format APDU ; le code programme réalisant les opérations de sérialisation dans la plate-forme embarquée étant mémorisé, en tant qu'implémentation d'au moins une méthode SendObject faisant appel à la méthode SendAPDU, dans une deuxième classe (ObjectIOApplet) de carte héritant de la classe (IOApplet).

Un autre but de l'invention est de proposer un système informatique incluant un tel objet portable et comportant de tels outils logiciels.

Cet objectif est atteint par un système informatique comprenant une station informatique, dite plate-forme embarquée, comprenant un objet portable incluant au moins un processeur, des moyens de mémorisation, et des moyens de communication aptes à échanger des informations avec un terminal sous la forme d'une ou plusieurs successions linéaires de données, caractérisé en ce que la plate-forme comprend un agent de sérialisation apte à effectuer une étape de conversion d'un ensemble de données, dans un sens ou dans l'autre, entre d'une part un agencement en une succession linéaire de données et d'autre part un agencement structuré décrivant ou représentant un ou plusieurs objets logiciels structurés ou hiérarchisés suivant les critères d'un langage de programmation orienté objet.

Selon une particularité, la plate-forme embarquée comprend un agent de communication apte à :

- recevoir en lieu et place de l'agent destinataire un ensemble de données reçues par la fonction de réponse à l'intention d'un agent logiciel destinataire mémorisé dans la plate-forme embarquée, cet ensemble de données étant agencé en une ou plusieurs successions linéaires de données ;
- convertir cet ensemble de données en au moins un objet logiciel, structuré ou hiérarchisé suivant les critères d'un langage de programmation orienté objet ;
- transmettre cet objet logiciel structuré à l'agent destinataire et déclencher un traitement en fonction de cet objet par cet agent destinataire.

Selon une particularité, la succession linéaire de données représentant un objet logiciel structuré est mémorisée dans la plate-forme embarquée dans un flux d'entrée ou un flux de sortie, la plate-forme embarquée comportant un agent logiciel dit agent de sérialisation apte à créer dans la plate-forme embarquée le ou les objets structurés représentés par le flux d'entrée, c'est à dire à désérialiser ces objets structurés, ou à écrire dans le flux de sortie des données représentant le ou les objets structurés à transmettre, c'est à dire à sérialiser ces objets structurés.

Selon une particularité, le flux d'entrée ou le flux de sortie sont mémorisés sous la forme d'une ou plusieurs structures circulaires de mémoire.

Selon une particularité, l'agent de sérialisation utilise une pile mémoire, dite pile de types, pour mémoriser le type d'au moins un objet composant tout ou partie de la structure d'un objet structuré à sérialiser ou désérialiser, cette pile de type comportant des emplacements mémoires qui ne sont chacun accessible qu'après que les emplacements mémorisés plus récemment aient été lus et effacés.

Selon une particularité, les données contenues dans les flux d'entrée ou de sortie représentent un ou plusieurs objets structurés en utilisant un codage comprenant un jeu de balises, ces balises représentant chacune une action déterminée à effectuer lors de la désérialisation de cette succession linéaire de données.

Selon une particularité, au moins une balise est définie comme représentant une des actions suivantes :

- ajout d'un nouvel élément à la structure de l'objet structuré représenté par la succession linéaire de données ;
- référence à un élément ou objet, dit objet source, en tant que source de la valeur de tout ou partie d'un élément composant l'objet structuré ;
- indication que la ou les données suivantes représentent le contenu d'un élément composant l'objet structuré ;
- indication d'une absence de contenu d'un élément composant l'objet structuré.

Selon une particularité, la plate-forme embarquée comprend un objet portable fonctionnant selon la norme ISO7816 et utilisant des commandes au format APDU.

Selon une particularité, au moins un agent ou application mémorisé dans la plate-forme embarquée est programmé en langage Java®, la plate-forme embarquée comportant un environnement informatique selon le standard JavaCard®.

Selon une particularité, le système comporte dans l'hôte ou dans la plate-forme embarquée, ou les deux, au moins une classe logicielle implémentant au moins l'une des commandes suivantes :

- une commande d'écriture d'objet, effectuant la transmission d'un objet structuré à au moins un agent de la carte, par sérialisation de cet objet structuré en un flux de données dans l'hôte, puis envoi de ce flux de données dans la plate-forme embarquée, puis dé-sérialisation de ce flux de données en un objet structuré dans la plate-forme embarquée ;
- une commande de lecture d'objet, effectuant la lecture d'un objet structuré depuis au moins un agent de la carte, par sérialisation de cet objet structuré en un flux de données dans la plate-forme embarquée, puis réception depuis la plate-forme embarquée de ce flux de données, puis désérialisation de ce flux de données en un objet structuré dans l'hôte ;
- une commande de désallocation d'un objet structuré mémorisé dans la plate-forme embarquée, par une sérialisation de cet objet selon une option comprenant une libération ou désallocation de l'espace mémoire alloué à chaque composant de cet objet, après analyse de la structure de ce composant ;
- une commande de nettoyage ou effacement d'un espace mémoire libéré par un objet structuré dans la plate-forme embarquée, par désérialisation d'une succession linéaire de données déterminée pour créer un objet de contenu sans signification ;
- une commande de duplication dans la plate-forme embarquée d'un objet structuré dit de départ, par sérialisation en une succession linéaire de données représentant ce même objet, sans désallocation de cet objet de départ, puis par désérialisation à partir de cette succession linéaire de données en un autre objet structuré de contenu identique à l'objet de départ.

Selon une particularité, la plate-forme embarquée communique au moins un hôte appartenant à un réseau informatique communiquant par messages asynchrones selon une infrastructure logicielle de type AAA-MOM, cet hôte comprenant un agent logiciel, dit agent proxy de moteur de carte, apte

à gérer les communications de cette plate-forme embarquée avec d'autres agents de ce réseau, ces agents fonctionnant selon les spécifications de cette infrastructure logicielle et appartenant à au moins une application distribuée.

L'invention, avec ses caractéristiques et avantages, ressortira plus clairement à la lecture de la description faite en référence aux dessins annexés dans lesquels :

- la figure 1 représente un schéma symbolique partiel des transferts d'objets et de leurs conversions lors d'opérations d'écriture et de lecture d'un hôte ou terminal vers et depuis un agent logiciel d'une plate-forme embarquée, dans un mode de réalisation de l'invention où la carte ne comporte qu'une seule application ;
- la figure 2 représente un schéma symbolique plus détaillé des transferts d'objets et de leurs conversions lors d'opérations d'écriture et de lecture d'un hôte ou terminal vers et depuis un agent logiciel d'une plate-forme embarquée, dans un mode de réalisation de l'invention où la carte ne comporte qu'une seule application ;
- la figure 3 représente un schéma symbolique partiel des transferts d'objets et de leurs conversions lors d'opérations d'écriture et de lecture d'un hôte ou terminal vers et depuis un agent logiciel d'une plate-forme embarquée, dans un mode de réalisation de l'invention où la carte comporte plusieurs applications ;
- la figure 4 représente un schéma symbolique partiel des objets et agents mis en œuvre lors de la désérialisation d'un objet logiciel structuré, à partir d'un flux de données ;

- la figure 5 représente un schéma symbolique partiel des objets et agents mis en œuvre lors de la sérialisation d'un objet logiciel structuré, vers un flux de données.

Dans certains ouvrages (par exemple « Java embarqué » - Eyrolles – Paris 1999), le terme d'ordinateur « embarqué » est utilisé pour désigner un ordinateur qui n'est pas visible en tant que tel mais intégré dans un équipement doté d'une autre fonction. Ce terme serait une traduction approximative de l'anglais « embedded computer », qui signifie littéralement « ordinateur enfoui » ou « enchâssé ».

Cette caractéristique, de remplir une fonction particulière au sein d'un autre dispositif, explique grandement le fait que de tels ordinateurs embarqués disposent souvent de ressources matérielles ou logicielles limitées, et utilisent souvent un système d'exploitation ou un environnement logiciel rudimentaire. Bien que des évolutions soient prévisibles, les ressources de tels ordinateurs peuvent typiquement être de l'ordre de 512 kilo-octets de mémoire statique à quelques kilo-octets, pour un processeur de 8 ou 16 bits. Dans le cas d'une carte à puce, la mémoire vive (RAM) disponible peut être d'environ 4 kilo-octets, et aller actuellement jusqu'à 32 kilo-octets pour les modèles les plus performants.

De façon générale en informatique, le terme de plate-forme désigne un dispositif de traitement de données comprenant au moins un processeur et des moyens de mémorisation. Par extension à la définition ci-dessus, et pour la présente description, le terme de « plate-forme embarquée » sera utilisé pour désigner un objet portable incluant une telle plate-forme, cet objet étant portable ou de faible taille, ou ayant de faibles capacités de traitement ou de mémorisation.

La présente description expose le procédé selon l'invention dans des modes de réalisation comportant une répartition déterminée des tâches et opérations entre les différents agents et applications concernés. La flexibilité de

l'organisation d'une application informatique permet bien sûr de présenter différemment cette répartition, en particulier lorsque les distinctions entre les différents agents et leurs dénominations sont des notions abstraites n'influant pas sur leurs caractéristiques principales de fonctionnement. Il est donc évident que le procédé selon l'invention peut également être mis en œuvre dans d'autres modes de réalisation non décrits ici, sans sortir de l'esprit de l'invention, en particulier en combinant différemment les diverses variantes exposées pour chaque tâche ou agent. De même, la répartition abstraite des tâches entre des agents ou applications mémorisés en un même lieu et temps, peut être combinée en diverses variantes non décrites ici sans sortir de l'esprit de l'invention.

Dans la présente description, le procédé selon l'invention sera illustré principalement dans le cas d'une plate-forme embarquée comprenant une carte à puce (souvent appelée « smartcard » en anglais) utilisant un environnement système de type JavaCard®, communiquant avec un terminal comprenant une station de traitement de données exécutant une application programmée en langage Java®.

Il doit toutefois être évident que le procédé selon l'invention peut être appliqué à d'autres environnements dont les fonctions de transmission de données ne permettent pas de transmettre des objets logiciels structurés de façon transparente pour le programmeur. Il peut ainsi s'agir de cartes à puce selon la norme ISO 7816 utilisant un autre environnement de programmation, par exemple « Windows for Smart Cards® », ou programmable avec un autre langage de programmation, par exemple Visual Basic®. Il peut également s'agir de cartes à puce selon une autre norme comportant des limites similaires dans leurs fonctions de communication, ou d'objets ou terminaux portables utilisant une telle plate-forme

De même, il peut aussi bien s'agir d'objets portables quelconques comprenant une station de traitement de données embarquée, amovible ou non, comme par exemple un composant électronique automobile, un marqueur de repérage, un téléphone cellulaire, ou un terminal de saisie portable.

De même, le procédé selon l'invention sera illustré principalement comme appliqué à l'utilisation d'une plate-forme embarquée communiquant avec une station de traitement de donnée, dite terminal ou hôte. Ce procédé peut bien sûr également être utilisé dans le cas ou le terminal est une station faisant partie d'un réseau informatique de quelque type que ce soit, sans sortir de l'esprit de l'invention. Ainsi, les différentes fonctions présentées comme réalisées par ce terminal peuvent aussi bien être réparties sur plusieurs dispositifs différents, voire selon une répartition variant dans le temps. Bien sûr, le procédé décrit peut également être appliqué au cas où la plate-forme embarquée communique directement ou indirectement avec une ou plusieurs autres plates-formes embarquées, sans sortir non plus de l'esprit de l'invention. De façon générale, l'hôte ou le terminal hôte sera donc défini comme l'application ou la station communiquant avec la plate-forme embarquée.

Le procédé selon l'invention peut en particulier être appliqué à la communication d'une carte à puce, par exemple au standard JavaCard®, avec un réseau informatique, par exemple programmé en Java®, communiquant par messages asynchrones selon une infrastructure logicielle de type AAA-MOM. Dans ce cas, les communications de la carte avec le reste du réseau sont typiquement gérées par un agent logiciel, dit agent proxy d'agent de carte, qui sert d'intermédiaire pour chacun des agents logiciels, dits agents de carte, mémorisés dans la carte. Dans l'environnement de la carte, ces agents de cartes sont gérés ou animés, ou les deux, par un agent logiciel dit agent moteur de carte.

Le procédé selon l'invention permet alors à cet agent moteur de carte de communiquer avec l'agent proxy de moteur de carte en s'échangeant des objets logiciels structurés selon les normes du langage Java ou de l'infrastructure AAA. Le fait de pouvoir échanger des objets structurés avec l'extérieur permet ainsi à la carte d'être mieux compatible avec l'infrastructure AAA, et d'être vue elle-même comme un agent de type AAA par le reste des agents AAA de ce réseau.

Le procédé selon l'invention peut également être appliqué à la communication d'une carte à puce, par exemple au standard JavaCard®, avec un réseau informatique, par exemple programmé en Java®, communiquant utilisant un protocole RPC (« Remote Procédure Call ») orienté objet, par exemple selon une infrastructure logicielle de type Java RMI (« Remote Method Invocation ») ou CORBA (marque déposée).

Lorsqu'une application nécessite de transférer des objets logiciels depuis un terminal vers une plate-forme embarquée constituée d'une carte à puce, ce transfert s'effectue souvent sous la forme d'une communication de type maître/esclave. C'est à dire que c'est le terminal qui prend l'initiative d'exécuter une fonction de transmission vers la plate-forme embarquée. Pour communiquer, cette fonction envoie à cette plate-forme une commande de communication assortie de paramètres d'envoi. Cette commande est alors capable de déclencher un ou plusieurs traitements au sein de la plate-forme, et de recevoir en retour des paramètres de retour ou de réponse.

Dans le cas d'une carte à puce fonctionnant selon la norme ISO 7816, les paramètres de cette commande sont transmis suivant un format de type APDU (« Application Protocol Data Unit »), qui consiste en une succession de données organisées comme suit :

Paramètres d'envoi transmis avec la commande :

en-tête				corps		
CLA	INS	P1	P2	Lc	Data T	Le

CLA : un octet désignant la classe ou l'application destinataire

INS : un octet désignant une instruction à exécuter

P1, P2 : deux octets apportant des précisions sur la commande APDU

Lc : longueur des données envoyées

Data : données envoyées

Le : longueur des données attendues en retour

Paramètres de réponse retournés par la carte :

corps	fin	
Data R	SW1	SW2

Data : données répondues

SW1, SW2 : deux octets constituant des messages ou codes de contrôle renvoyés par la carte

On voit que ce format APDU ne permet d'envoyer ou de recevoir de données que sous forme d'une succession linéaire de données, c'est à dire une simple suite d'octets. Lorsqu'une application est programmée dans un langage ou pour une plate-forme embarquée dont le système ne dispose que des commandes APDU pour communiquer, le programmeur qui veut transférer des objets plus structurés ne dispose donc pas d'outils logiciels simples. Il est obligé de prévoir dans son application de convertir de tels objets structurés en une suite d'octets, puis d'utiliser la commande APDU pour les transmettre, puis de reconvertir ces objets dans le sens inverse. Il s'agit là de réaliser « manuellement », c'est à dire de programmer dans les moindres détails, la sérialisation, transmission, puis désérialisation de ces objets.

Dans le cas d'un programmeur utilisant le langage Java® pour développer une application utilisant des plates-formes embarquées au standard JavaCard®, les outils disponibles dans JavaCard® consistent en une transmission de données linéaires au format APDU, avec les commandes JavaCard® suivantes :

- « process(APDU) » : la carte reçoit des données au format APDU et lance le traitement des paramètres d'envoi qu'elles contiennent, par l'agent logiciel, ou « applet », destinataire désigné dans ces données ;

- « sendAPDU() » ou « APDU.sendbytes() » : après ou au cours du traitement déclenché précédemment, la carte retourne vers l'hôte d'autres données au format APDU, contenant les paramètres de retour.

La commande « sendAPDU() » est implémentée au sein de l'environnement JavaCard dans la classe « IOApplet », sous la forme d'une méthode applicable à un objet non typé contenant des données brutes.

La commande « Process(APDU) », par contre, est déclarée au sein de l'environnement JavaCard dans la classe « IOApplet », sous la forme d'une méthode abstraite. C'est à dire que la méthode existe dans l'environnement Javacard, mais que le code réalisant son fonctionnement doit être écrit par le programmeur d'une application voulant utiliser cette commande. Le programmeur va par exemple créer dans son application une sous-classe héritant de la classe « IOApplet », cette sous-classe contenant alors une méthode recevant le code du traitement à exécuter par cette méthode « Process(APDU) ». L'environnement JavaCard se contente d'appeler la méthode « Process(APDU) » lors de la réception d'un message, et donc de lancer le traitement que le programmeur a prévu dans son code ajouté.

De façon à fournir à un tel programmeur des outils lui permettant de transmettre directement des objets logiciels structurés, le procédé selon l'invention fournit des commandes similaires mais acceptant directement des objets logiciels structurés, selon les spécificités orientées objet du langage Java®. Ces commandes sont alors utilisables dans l'environnement JavaCard® et peuvent être par exemple du type : « process(Object) » et « sendObject() ».

Pour permettre au programmeur, également ici désigné comme utilisateur, d'utiliser ces commandes directement sans se préoccuper de la syntaxe des commandes APDU, c'est le procédé selon l'invention qui va prendre en charge les opérations de sérialisation, transmission, et désérialisation des objets et données, entre l'application ou agent émetteur et l'agent destinataire.

Dans le cas d'un environnement JavaCard, ces opérations peuvent alors être réalisées par l'exécution d'un code programme contenu dans la méthode « Process(APDU) » appartenant à une sous-classe de la classe « IOApplet », par exemple nommée « ObjectIOApplet ». Ce code est donc automatiquement lancé par l'environnement lors de la réception d'un message APDU, et contient lui-même les opérations de conversions ainsi que l'appel d'une autre méthode abstraite « Process(Object) ». Grâce à un tel mode de réalisation de l'invention, le programmeur n'a plus qu'à écrire le code des traitements qu'il veut voir réalisés par la carte lors de la réception d'un objet structuré. Il va alors écrire ce code dans une sous-classe héritant de la classe « ObjectIOApplet », en tant qu'implémentation de la méthode abstraite « Process(Object) ».

Dans la présente description, seules les opérations de conversion, sérialisation ou désérialisation, effectuées du côté de la carte seront décrites. Il est bien sûr évident que le procédé décrit peut être utilisé aussi bien pour réaliser les mêmes opérations du côté de l'hôte. Les ressources matérielles et logicielles du terminal hôte étant le plus souvent bien supérieures à celles disponibles sur la carte, ces opérations pourront toutefois également être programmées ou organisées de façon différente du côté de l'hôte, sans sortir de l'esprit de l'invention.

Dans un mode de réalisation représenté en figure 1, un terminal hôte (1) communique avec une plate-forme embarquée, par exemple une carte à puce (2) au standard Javacard®. L'hôte (1) exécute au moins une application (11) comprenant au moins un agent logiciel (111), et communique avec la carte (2) par une fonction de communication (101) au format APDU mettant en œuvre des moyens de communication comportant par exemple un emplacement de connexion (100). Cet emplacement de connexion comporte des moyens de connexion, par exemple par contact électrique, par liaison hertzienne, par liaison infrarouge, par piste magnétique, ou une combinaison de plusieurs de ces types.

La carte (2) exécute une application (22) comprenant au moins un agent logiciel (221), et communique avec l'hôte (1) en utilisant une fonction de réponse (201) faisant partie de l'environnement système JavaCard®. Cette fonction de réponse (201) utilise pour cela des moyens de communication (200) d'un type compatible avec les moyens de communication (100) du terminal hôte (1).

Lorsque l'agent (111) de l'application (11) hôte veut envoyer des données à l'agent (221) de la carte, ou lui faire exécuter un traitement (2210), ou lui demander des informations qu'elle a en mémoire, il exécute une instruction d'écriture d'objet lançant le procédé dans le sens de l'envoi vers la carte d'un objet logiciel structuré (31) , cette instruction étant par exemple dénommée « WriteObject() ». Cet objet logiciel (31) est alors sérialisé, c'est à dire converti en un ensemble de données au format APDU, par un agent hôte de conversion (12). Cet agent hôte de conversion utilise alors la fonction de communication (101) pour transmettre ces données à la carte (2) à travers l'emplacement de connexion (100) de l'hôte et les moyens de communication (200) de la carte.

Une fois reçues dans la carte par la fonction de réponse (201) gérant les moyens de communication (200), cette fonction de réponse transmet ces données à un agent de conversion de carte (21). Cet agent de conversion de carte (21) désérialise ces données, c'est à dire les convertit dans le sens inverse de façon à leur redonner leur structure d'origine, sous la forme d'un objet logiciel (34). Cet objet logiciel structuré (34) est alors transmis à l'agent (22) destinataire grâce à une instruction de traitement d'objet qui va effectuer le traitement (2210) correspondant aux données reçues, cette instruction étant par exemple dénommée « Process(Object) ».

Lorsque l'agent (221) de l'application (22) de la carte (2) veut envoyer des données à l'agent (111) de l'hôte (1), ou des informations sur un traitement effectué, il exécute une instruction d'envoi d'objet lançant le procédé dans le sens de l'envoi vers l'hôte d'un objet logiciel structuré (41) , cette instruction étant par exemple dénommée « SendObject() ». Cet objet logiciel (41) est alors sérialisé, c'est à dire convertit en un ensemble de données au format APDU,

par exemple par l'agent de conversion de carte (21). Cet agent de conversion de carte utilise alors la fonction de réponse (201) pour renvoyer ces données à la carte (2) à travers les moyens de communication (200) de la carte et l'emplacement de connexion (100) de l'hôte.

Une fois reçues dans l'hôte par la fonction de communication (101) gérant l'emplacement de connexion (100), cette fonction de communication transmet ces données à l'agent hôte de conversion (12). Cet agent hôte de conversion (12) désérialise ces données, c'est à dire les converti dans le sens inverse de façon à leur redonner leur structure d'origine, sous la forme d'un objet logiciel (44). Cet objet logiciel structuré (44) est alors transmis à l'agent (11) destinataire par une instruction de lecture d'objet qui va lire les données reçues, cette instruction étant par exemple dénommée « ReadObject() ».

La figure 2 représente un mode de réalisation de l'invention où les opérations de sérialisation/désérialisation et transmission réalisées par les agents (12, 21) de conversion respectivement d'hôte et de carte sont effectuées par deux agents différents (121, 122, respectivement 211, 212).

L'agent (21) de conversion de carte comprend ainsi un agent (211) de communication et un agent (212) de sérialisation. L'agent (211) de communication gère les opérations de conversion au niveau de l'octet, en échangeant des données sous forme de paramètres d'envoi (32) et de réponse (43) avec la fonction (201) de réponse. En réception de données, cet agent (211) de communication vérifie que les données reçues sont complètes, et effectue une concaténation des paramètres d'envoi (32) en une succession linéaire de données, mémorisée dans un flux (33) d'entrée. En envoi de données, cet agent (211) de communication lit une succession linéaire de données dans un flux (42) de sortie et sépare ces données pour les répartir en paramètres (43) de réponses, compatibles avec les possibilités de transmission de la fonction (201) de réponse.

L'agent (212) de sérialisation gère les opérations de conversion au niveau de l'objet, en réalisant la sérialisation/désérialisation proprement dite.

Lors d'un envoi de données, donc en sérialisation, l'agent (212) de sérialisation, analyse la structure et le contenu d'un objet (41) logiciel structuré à transmettre, et réalise un codage de cet objet sous la forme d'une succession linéaire de données, mémorisée dans le flux (42) de sortie. Lors d'une réception de données, donc en désérialisation, l'agent (212) de sérialisation lit une succession linéaire de données dans le flux (33) d'entrée. Il analyse alors les données qui s'y trouvent et reconstitue l'objet (34) logiciel structuré qu'elles représentent.

La figure 3 représente un mode de réalisation de l'invention, où la carte peut comporter plusieurs agents (221, 222, 223, 231) susceptibles d'être destinataire pour le procédé selon l'invention, ces agents pouvant être répartis dans une ou plusieurs applications (22, 23). De façon à pouvoir traiter tous les échanges de données entre la carte (2) et l'hôte (1), le procédé selon l'invention prévoit un agent (210) d'interposition par qui transitent tous les échanges de données structurées entre la carte (2) et l'hôte (1).

Dans ce mode de réalisation, l'agent (210) d'interposition reçoit toutes les données transmises à la carte, en lieu et place de l'application (22) ou l'agent (221) destinataire, et quel que soit ce destinataire. Cet agent (210) d'interposition va alors faire effectuer la concaténation des paramètres (32) d'envoi par l'agent (211) de communication, et faire ensuite désérialiser les données (33) résultantes par l'agent (212) de sérialisation, comme décrit ci-dessus. Une fois ces données reconstituées en un objet (34) logiciel structuré, c'est ce même agent (210) d'interposition qui va transmettre cet objet (34) structuré à l'agent (221) logiciel qui était destinataire des données (32) lors de leur réception. Dans un mode de réalisation (non représenté), les données (32) reçues par l'agent (210) d'interposition contiennent une information représentant l'identification de l'agent (221) destinataire, ou bien l'agent (210) d'interposition ajoute une telle information à ces mêmes données (32) avant de les transmettre. L'objet (34) structuré résultant de la désérialisation est alors directement adressé par l'agent (212) de désérialisation ou par un agent de distribution (non représenté).

De la même façon, l'agent (210) d'interposition reçoit toutes les données (41) envoyées vers l'hôte par une application (22) ou un agent (221) émetteur. Cet agent (210) d'interposition va alors faire sérialiser ces données par l'agent (212) de sérialisation en un flux (42) de sortie, puis faire concaténer ce flux (42) de sortie par l'agent (211) de communication, comme décrit ci-dessus. Les données obtenues seront alors envoyées par la fonction (201) de réponse vers le terminal (1) à travers les moyens de communication (200, 100) sous la forme de paramètres (43) de réponse.

Dans un mode de réalisation (non représenté), l'agent (210) d'interposition s'intercale au sein de la chaîne (201, 211, 212, 221) de transmission et de conversion. En réception dans la carte (commande d'écriture de la part de l'hôte), l'agent (210) d'interposition dirige les données ou objets reçus vers leur destinataire (221, 222, 223, 231) au sein de la carte à partir d'informations inclues dans les données reçues. En émission depuis la carte (commande de lecture de la part de l'hôte), l'agent (210) d'interposition peut également être prévu pour recevoir les données ou objets à émettre et leur affecte une information représentant leur émetteur (221, 222, 223, 231).

Entre d'une part l'hôte (1), que ce soit le terminal lui-même ou un agent logiciel quelconque capable de gérer ce terminal, et d'autre part un agent ou une application présent sur la carte, on peut considérer que les échanges d'informations se font donc sur deux niveaux différents : au niveau de l'octet d'une part, et au niveau de l'objet d'autre part.

Au niveau des objets, les données organisées en objets logiciels structurés à transmettre sont converties en un flux linéaire de données et réciproquement. Cette étape de sérialisation est celle qui gère la structure des objets et est réalisée à l'intérieur de chaque plate-forme par l'agent de sérialisation.

Au niveau des octets, ou suites d'octets, les données agencées en simples flux linéaires sont transmises par bribes par l'intermédiaire de la fonction de transmission, par exemple au format APDU, qui est le seul moyen d'échange d'informations entre la carte et le monde extérieur. Ces échanges

sont gérés par les agents de communication hôte et de la carte, qui communiquent entre eux en s'envoyant des paramètres grâce à cette fonction de transmission.

De façon à fournir à l'utilisateur des commandes transparentes, les différents processus sont implémentés dans une ou plusieurs classes, au moins une fonctionnant dans la carte et une fonctionnant dans le terminal ou l'hôte en général. Dans le langage Java®, le procédé selon l'invention peut ainsi fournir une classe « ObjectIOApplet » pour la carte et une classe « ObjectIOProxy » pour l'hôte, par exemple selon la syntaxe suivante :

```
class ObjectIOProxy {
    void      writeObject (Object o) ;
    Object    readObject () ;
}
```

La classe « ObjectIOProxy » définissant une classe dans l'hôte, fournissant à l'application de l'utilisateur les commandes d'écriture d'objet « writeObject (Object) » et de lecture d'objet « readObject () ».

```
class ObjectIOApplet {
    void      process (Object o) ;
    void      sendObject (Object o) ;
}
```

La classe « ObjectIOApplet » définissant une classe dans la carte, fournissant à l'application de l'utilisateur les commandes de traitement d'objet « process(Object) » et d'envoi d'objet « sendObject() » ; et

La carte étant essentiellement passive, l'ensemble de ces conversions et transmissions se fait sur initiative de l'hôte, par une instruction dans le code déclenchant l'envoi d'une commande APDU. C'est cette commande qui va déclencher l'opération de conversion demandée auprès de l'agent concerné.

Les commandes « process(Object) » et « sendObject() », lorsqu'elles sont exécutées dans le code de l'application de carte, ne déclenchent donc pas directement les opérations de conversion correspondantes. La commande « sendObject() » va mémoriser l'objet à envoyer dans une queue ou file de sortie, par exemple « qout », où les objets à sérialiser pour transmission seront

introduits par une commande d'introduction du type « qout.push(object) ». Lors du déclenchement d'une opération de sérialisation, ces objets à transmettre seront alors extraits de cette file de sortie dans l'ordre où ils y ont été introduits.

De même, la commande va extraire d'une file d'entrée, par exemple « qin », un objet sérialisé lors d'une opération précédente déclenchée par l'hôte. Cette extraction se fera alors par une commande d'extraction du type « qin.pop() ».

Les opérations de lecture, écriture, sérialisation et désérialisation sont alors gérées depuis l'hôte par la classe « ObjectIOProxy », grâce à des commandes transparentes à l'utilisateur. Ces commandes peuvent être implémentées dans le code de cette classe, par exemple sous la forme suivante :

- « card.Serialize out » : lance dans la carte la sérialisation, comme décrit ci-dessous, des objets à envoyer. Ces objets peuvent avoir été mémorisés dans une file d'attente de sortie lors de l'exécution d'une instruction sendObject() par l'application de la carte.
- « card.Read » : lance dans la carte la lecture des données du flux de sortie et leur transmission vers l'hôte, comme décrit ci-dessous.
- « card.Write » : lance l'envoi de données vers la carte et leur écriture dans le flux d'entrée, comme décrit ci-dessous.
- « card.Serialize In » : lance dans la carte la dé-sérialisation, comme décrit ci-dessous, des objets présents dans le flux d'entrée.

Dans le mode de réalisation décrit ici, le flux de sortie et le flux d'entrée de la carte sont mémorisés dans une même structure mémoire circulaire. D'autres modes de réalisation sont bien sûr possibles utilisant une structure mémoire différente pour chaque flux, ou utilisant d'autres types de structures mémoires.

Une structure de mémoire circulaire signifie en l'occurrence un ensemble d'emplacements de mémoire se suivant, dont le premier emplacement est considéré par le système comme étant immédiatement à la

suite du dernier emplacement. C'est à dire qu'une succession linéaire de données peut être mémorisée dans une telle structure en commençant à n'importe quel emplacement sans perdre sa continuité, du moment que la longueur de cette succession n'est pas supérieure au nombre total d'emplacements.

Le fait que les flux d'entrée et de sortie partagent la même structure circulaire consiste à mémoriser les données qu'ils contiennent dans des zones différentes de cette même structure circulaire. Les données contenues dans ces flux étant effacées au fur et à mesure de leur lecture, immédiatement ou lors d'une opération, la longueur et la position de ces deux flux varient à l'intérieur de cette structure circulaire. Lorsque l'un des deux flux n'a plus de place pour mémoriser de nouvelles données parce qu'il est bloqué par les données non effacées de l'autre flux, il suffit alors de traiter cet autre flux pour libérer la place qu'il occupe.

Une telle structure circulaire permet de n'utiliser que peu de ressources mémoire tout en traitant des flux d'une longueur non déterminée à l'avance. Il suffit pour cela que les différentes opérations agissant sur ces deux flux s'intercalent de façon suffisamment équilibrée.

Au niveau des octets, depuis le flux de sortie d'une plate-forme vers le flux d'entrée de l'autre plate-forme, et réciproquement, les échanges de données se font de la façon suivante.

Lorsque l'hôte veut obtenir des données se trouvant dans le flux de sortie de la carte, il émet une commande de lecture, par exemple selon la syntaxe Java® : « card.Read ». Cette commande lance une session de lecture, qui est typiquement divisée en plusieurs transactions, représentant chacune une itération du processus.

Pour initier la session de lecture, à travers son agent de communication et sa fonction de transmission, l'hôte émet une commande au format APDU, assortie de paramètres d'envoi (P1 et P2, soit un entier court) représentant la longueur de données qu'il veut recevoir. A réception de cette commande, de

l'agent de communication de carte lit un premier bloc de données dans le flux de sortie de la carte et le transmet à la fonction de réponse de la carte. Ce premier bloc de données est alors retourné à l'hôte en tant que données retournées (DataR) dans la réponse à la commande APDU.

Chaque itération successive du processus de lecture comprend alors l'envoi par l'hôte d'une commande APDU assortie de paramètres d'envoi (P1 et P2, soit un entier court) représentant la longueur des données déjà reçues. Une fois reçue par l'agent de communication de carte, cette longueur fait office d'accusé de réception pour les envois précédents. Ainsi, l'agent de communication retourne comme données de réponse (DataR) les données du flux de sortie suivant immédiatement la longueur indiquée par l'hôte. Il est à noter que la lecture des données dans le flux de sortie se fait sans effacement de celles-ci, ce qui permet leur ré-envoi en cas de besoin. On comprend bien qu'ainsi aucune donnée ne peut être oubliée dans cette transmission.

La session de lecture se termine lorsque la longueur demandée par l'hôte a été bien reçue, par arrêt des itérations de la part de l'hôte. La session peut également se terminer, ou s'interrompre, si l'agent de communication de carte renvoie une donnée (par exemple un champ DataR de longueur nulle) ou un code (SW1 ou SW2) signifiant que toutes les données disponibles dans le flux de sortie ont déjà été envoyées. Dans le cas d'un flux de sortie vide dans la carte, il incombe alors à l'hôte de déclencher dans la carte une nouvelle opération de sérialisation d'objets depuis la file de sortie « qout » de la carte vers le flux de sortie de la carte, pour que ce flux de sortie reçoive de nouvelles données.

Lorsque l'hôte veut envoyer à la carte des données se trouvant, il émet une commande d'écriture, par exemple selon la syntaxe Java® : « card.Write ». Cette commande lance une session d'écriture, qui est typiquement divisée en plusieurs transactions, représentant chacune une itération du processus.

Pour initier la session d'écriture, à travers son agent de communication et sa fonction de transmission, l'hôte émet une commande au format APDU,

assortie de paramètres d'envoi (P1 et P2, soit un entier court) représentant la longueur des données qu'il veut transmettre. Le champ de données (DataT) de cette première commande APDU contient alors le premier groupe ou bloc de données à envoyer, ce bloc de données étant lu dans le flux de sortie de l'hôte. A réception de cette commande, la fonction de réponse de la carte transmet ce premier bloc de données à l'agent de communication de carte. L'agent de communication de carte écrit alors ce premier bloc de données dans le flux d'entrée de la carte.

Chaque itération successive du processus d'écriture comprend alors l'envoi par l'hôte d'une commande APDU assortie de paramètres d'envoi (P1 et P2, soit un entier court) représentant la longueur des données déjà envoyées. Le champ de données (DataT) de cette première commande APDU contient alors le groupe ou bloc de données suivant, lu dans le flux de sortie de l'hôte. A réception de cette commande, la fonction de réponse de la carte transmet ces paramètres et ce bloc de données à l'agent de communication de carte. L'agent de communication de carte compare alors la longueur de données annoncées dans les paramètres d'envoi (P1 et P2) avec la longueur des données déjà reçues depuis le début de la session d'écriture. On comprend bien qu'ainsi aucune donnée ne peut être oubliée dans cette transmission.

Si cette comparaison ne relève pas d'erreur, l'agent de communication de carte écrit alors ce bloc de données dans le flux d'entrée de la carte. En cas d'erreur, l'agent de communication retourne à l'hôte un code ou un indice indiquant une erreur et/ou représentant la nature de cette erreur. Ce code ou indice peut être retourné à travers les paramètres de réponse (SW1 et SW2) de la fonction de réponse, ou à travers le champ de données retournées (DataR) ou la longueur de ce champ, ou une combinaison de ces éléments.

La session d'écriture se termine lorsque la longueur annoncée par l'hôte a été bien transmise, par arrêt des itérations de la part de l'hôte. La session peut également se terminer, ou s'interrompre, si l'agent de communication de carte renvoie un code ou indice signifiant que le flux d'entrée de la carte ne peut plus accueillir de nouvelles données. Il incombe

alors à l'hôte de déclencher dans la carte une ou plusieurs opérations permettant de libérer de l'espace dans la structure mémoire qui contient ce flux d'entrée.

Il peut s'agir de déclencher dans la carte une nouvelle opération de désérialisation de données depuis le flux d'entrée vers la file d'entrée « qin » d'objets accessibles à l'application carte. Il peut s'agir également de déclencher une nouvelle session de lecture pour recevoir des données contenues dans le flux de sortie de la carte, et libérer ainsi de l'espace dans la structure circulaire qui contient ces deux flux.

Au niveau des objets, les opérations de sérialisation et désérialisation se font entre les flux de données et les agents ou applications de la carte, et réciproquement, de la façon suivante.

Au sein de la carte, les objets sont désérialisés depuis le flux (33) d'entrée vers la file d'entrée « qin » où ils sont directement accessibles à l'instruction « Process(Object) », fournie par la classe « ObjectIOApplet » et utilisée par l'agent ou l'application destinataire.

Dans l'implémentation du procédé selon l'invention, lorsque l'hôte veut déclencher dans la carte une opération de désérialisation, il utilise une instruction, par exemple selon la syntaxe Java® « card.Serialize In ».

Dans la figure 4 est illustrée la désérialisation d'un objet (34) logiciel structuré, à travers le décodage des données du flux (32) d'entrée correspondant à cet objet (34), et les opérations de mémorisation et de création des composants de la structure de ce même objet (34) structuré, ou objet résultat.

En lisant l'une après l'autre les données mémorisées en une succession linéaire dans le flux (33) d'entrée, l'agent (212) de sérialisation interprète ces mêmes données selon un codage déterminé et crée un objet (34) logiciel structuré en fonction de cette interprétation. Dans cette succession de données mémorisée dans ce flux d'entrée, certaines données peuvent avoir une valeur déterminée qui sera interprétée comme marquant la

présence d'une balise de codage. Ce décodage peut se faire par appel d'une méthode d'allocation propre à chaque type d'objet structuré à décoder, par exemple selon la syntaxe suivante :

```
Typeobject::decode (Object, InputStream)
```

pour le décodage d'un objet « Object » de type « Typeobject » à partir du flux d'entrée « Input Stream ».

Ce codage comprend un jeu de balises ayant chacune une signification déterminée, et correspondant chacune à au moins une valeur spécifique d'une donnée dans le flux d'entrée. Ainsi, dans le flux d'entrée, chaque donnée d'une valeur correspondant à une de ces balises est interprétée par l'agent de sérialisation lors du processus de désérialisation.

Selon les applications du procédé selon l'invention, l'agent (212) de sérialisation pourra être prévu pour reconnaître plusieurs jeux de balises, ou des balises pouvant être représentées chacune par plusieurs valeurs différentes de données, sans sortir de l'esprit de l'invention. Une telle diversité pourra en particulier être utilisée pour permettre une compatibilité d'une même carte (2) ou type de carte avec plusieurs terminaux ou environnements hôtes différents.

Dans le mode de réalisation décrit ici, le codage comprend des balises de type « NULL », « NEW », « REF » et « DATA ».

- Une balise NULL représente une absence de donnée, tout en occupant un emplacement dans la structure en cours de construction par l'agent sérialisation.
- Une balise NEW indique à l'agent de sérialisation le début de la description d'un nouvel objet, cet objet pouvant être soit un objet (34) logiciel structuré résultant de la conversion, soit un objet, dit élément, composant une partie d'un tel objet (34) structuré.
- Une balise REF indique la désignation d'un objet, un autre ou le même, en tant que source de la valeur de tout ou partie de l'objet ou élément en cours de description. Il s'agit d'affecter une valeur par référence.
- Une balise DATA indique que les données suivantes représentent la valeur, ou contenu, de l'objet ou élément en cours de description. Ce contenu peut

être constitué de données brutes représentant directement une ou des valeurs à affecter à cet objet, ou inclure d'autres balises indiquant à nouveau des objets ou références définissant ce contenu.

Selon son type, une balise peut être suivie d'une ou plusieurs données, dont l'interprétation sera déterminée par la signification de cette même balise.

- Ainsi, à la lecture d'une balise NEW, l'agent de sérialisation saura qu'il doit interpréter la donnée suivante comme représentant le type du nouvel objet décrit.
- De même, à la lecture d'une balise REF, l'agent de sérialisation saura qu'il doit interpréter la donnée suivante comme représentant un identifiant de l'objet désigné comme source de cette valeur par référence.

Dans le mode de réalisation décrit ici, les balises comme les identificateurs de type et de référence sont codés sur un octet, mais il est évident que le procédé selon l'invention permet aussi bien d'utiliser d'autres codages, différents voire plus complexes ou explicites, en fonction des besoins et des possibilités de l'environnement informatique concerné.

Au fur et à mesure de sa lecture des données du flux (32) d'entrée, l'agent (212) de sérialisation analyse la valeur de chacune de ces données, et effectue la création puis le remplissage des objets ou éléments (340, 341, 342, 343, 344) composant l'objet (34) résultat. Cette lecture du flux (32) d'entrée se répètera alors jusqu'à la reconstruction complète de l'objet (34) représenté par ce flux d'entré. Cette création peut se faire par appel d'une méthode d'allocation propre à chaque type d'objet, par exemple selon la syntaxe suivante :

```
Typeobject::malloc()
```

pour la création ou allocation d'un objet de type « Typeobject » lors de la désérialisation.

Du fait que de tels objets peuvent présenter une structure différente, la gestion de cette création et de ce remplissage se font, directement ou non, par un agent (TM0, TM1, TM2) de contrôle de type spécifique au type de l'objet à créer, par exemple un agent « type marshaller » du langage Java®. Cet agent de contrôle de type est spécifique à un ou plusieurs types d'objets, et peut être

géré par un agent (TMG) gestionnaire de types, par exemple un agent « type manager » du langage Java®. Cet agent gestionnaire de types mémorise en particulier les identifiants des différents types d'objets créés au cours du processus de désérialisation. Cet agent (TMG) gestionnaire de types comprend également le code et les procédures, ou méthodes, spécifiques à ces différents types d'objets et utilisés pour sérialiser/désérialiser ces mêmes objets. C'est également lui qui gère la liste des objets et leurs allocations, ce qui permet de réaliser de nouvelles allocations de réutiliser une allocation libre pour créer un nouvel objet lors du décodage.

De façon typique, cet agent (TMG) gestionnaire de types comprend des informations sur tous les différents types d'objets pouvant être gérés dans la carte. Ces informations peuvent être générées par un hôte, par exemple lors d'une phase de programmation de la carte. Elles sont alors transmises avec les classes (« applets ») utilisées par les applications, sous forme de code programme (« glue code ») s'ajoutant au code de l'agent (TMG) gestionnaire de types.

Par l'utilisation d'un tel agent gestionnaire de type, ainsi que d'agents de contrôle de type décrits ci-dessous, le procédé selon l'invention permet de gérer la sérialisation et la désérialisation d'objets structurés, de types de base ou de types construits, dans une plate-forme embarquée dont l'environnement de programmation, comme JavaCard par exemple, ne comporte pas un tel gestionnaire de type.

Au cours du décodage, la création d'un objet ou élément correspond à une allocation de cet objet, c'est à dire la réservation d'un espace mémoire déterminé et l'attribution de cet espace à cet objet. Certains environnements embarqués, et en particulier Javacard®, ne disposent pas d'outils logiciels permettant de rendre à nouveau disponible l'espace mémoire auparavant occupé par un objet qui a été supprimé, comme par exemple un agent « garbage collector » du langage Java® classique. Lors de la création d'un objet par l'agent (212) de sérialisation, le procédé selon l'invention peut prévoir

la possibilité de réaliser cette création en allouant à cet objet un espace mémoire précédemment occupé par un autre objet devenu inutile, par exemple de même type que l'objet à créer. Il est ainsi possible d'une fois sur l'autre de réutiliser l'espace mémoire de la carte, qui est une ressource précieuse dans de nombreux cas, car peu abondante.

Dans l'exemple illustré en figure 4, le flux (32) d'entrée correspond à un objet (34) structuré résultat dont la structure, ou graphe, peut être décrite en langage Java® sous la forme suivante :

`Type1 : {bool}`	`class B {boolean bo ;}`

`Type2 : Type1+{int, byte, Type0}`	`class X extends B {`
	`int i ;`
	`byte by ;`
	`Y y ;`
	`}`

`Type0 : {Type1 }`	`class Y { B b ;}`

Lors de la lecture du flux (33) d'entrée, l'agent de sérialisation lit tout d'abord une balise (321) NEW. Il lit donc la donnée (322) suivante et mémorise alors la demande de création d'un objet de type 2. Conformément à cette balise (321) NEW d'identifiant 2, l'agent de sérialisation va créer un nouvel objet, « x » (340) de classe X dans l'exemple, par l'intermédiaire de l'agent (TM2) de contrôle de type correspondant à ce même type (Type2). S'agissant du début de la description d'un objet résultat, ce nouvel objet sera l'objet « racine » de l'arborescence du graphe de cet objet résultat.

Lors de cette création, l'agent de sérialisation va attribuer à l'objet (340) créé un index (3402), cet index valant 0 dans l'exemple. Cet index (3402) peut ainsi servir de référence à d'autres éléments ou parties d'éléments lors de la construction de cet objet (34) résultat, et permettre de savoir si le contenu de cet élément à ou non été rempli.

L'agent de sérialisation va alors mémoriser le type et l'index de cet objet dans une structure mémoire, dite pile de types (TYST). Cette pile de types

est une structure de type pile mémoire, c'est à dire où l'on peut mémoriser (action dite « push » en anglais) des données l'une sur l'autre, et dont on ne peut lire et extraire (action dite « pop » en anglais) une donnée que lorsque ceux mémorisés après lui ont déjà été extraits. Les données quittent cette pile dans l'ordre inverse de celui où elles y ont été mémorisées (« dernier entré premier sorti », ou « LIFO » pour « Last In First Out »).

L'agent de sérialisation va également mémoriser le type de ce nouvel objet (340) ainsi que son index, comme correspondant à l'objet, dit objet (OBJ) en cours, qui sera rempli par les données suivantes du flux d'entrée.

La balise suivante est une balise DATA suivie de deux données (324, 325) brutes, qui ne sont pas des balises ni des identifiants. Ces deux données vont donc être mémorisées dans l'objet (OBJ) en cours, c'est à dire x (340), en tant que valeur des éléments (342, 343) suivants. Ces deux éléments suivants étant respectivement de types entier (int) dénommé « i » et octet (byte) dénommé « by », ces éléments vont donc prendre respectivement les valeurs contenues dans ces deux données (324, 325) du flux d'entrée.

La balise suivante est une balise NEW, suivie d'une donnée indiquant un type 0. Cette séquence indique que l'élément suivant de l'objet « x » est un objet de type 0. L'agent de sérialisation va donc affecter à cet objet (344) un index (3442), 1 dans l'exemple, et ajouter (« push ») l'index et le type, à savoir 0, de ce nouvel objet (344) sur la pile (TYST) de types. L'agent de sérialisation va également faire créer un objet de type 0, dans l'exemple « y » (344) de type Y, par l'agent (TM0) de contrôle de type correspondant au type 0.

A travers l'agent (TM2) de contrôle de type correspondant à l'objet (OBJ) en cours, l'agent de sérialisation sait que cet objet (OBJ) en cours, à savoir « x », n'est pas rempli complètement. La balise suivante, une balise DATA suivie d'une donnée (329) brute, sera donc affectée comme valeur de l'élément (341) suivant de l'objet en cours, c'est à dire comme valeur de l'élément dénommé « bo » et de type booléen qui est le dernier élément de l'objet « x ».

Maintenant, à travers l'agent (TM2) de contrôle de type correspondant à l'objet (OBJ) en cours, l'agent de sérialisation sait que cet objet (OBJ) en cours, à savoir « x » (340), est complètement rempli. L'index (3402) de cet objet (340) rempli est donc mémorisé dans la pile (OBJST) d'objets, avec l'identifiant de cet objet (340) au sein de la carte. L'agent de sérialisation va alors clore le remplissage de l'objet en cours et dépiler, ou extraire (« pop »), le type et l'index mémorisés sur le dessus de la pile (TYST) de types. Le dessus de la pile doit bien sûr être compris comme l'emplacement de cette pile accessible en premier. Le type et l'index extrait de la pile de types, 0 et respectivement 1 dans l'exemple, vont alors être mémorisés comme correspondant à un nouvel objet (OBJ) en cours.

La balise suivante est une balise DATA, qui indique donc un remplissage de l'objet en cours, c'est à dire « y ». Cette balise est suivie d'une balise (3211) REF et d'une donnée (3312), 0 dans l'exemple. L'agent de sérialisation va donc affecter à l'objet « y » (344) une valeur par référence, c'est à dire une simple liaison avec la valeur de l'objet dont l'index (3402) est désigné par cette référence (3212). L'objet « y » aura donc une valeur définie comme faisant référence à l'objet « x » qui porte l'index 0 au sein du processus de désérialisation. Cette liaison pourra alors être définie dans l'objet (34) résultat par la mémorisation de l'identifiant de cet objet « x » dans la carte, cet identifiant étant lu dans la pile (OBJST) d'objets grâce à cet index.

Maintenant, à travers l'agent (TM0) de contrôle de type correspondant à l'objet (OBJ) en cours, l'agent de sérialisation sait que cet objet (OBJ) en cours, à savoir « y », est complètement rempli. L'index (3442) de cet objet (344) rempli est donc mémorisé dans la pile (OBJST) d'objets, avec l'identifiant de cet objet (344) au sein de la carte. L'agent de sérialisation va alors clore ce remplissage et consulter la pile (TYST) de types.

Du fait que la pile (TYST) de types est vide après extraction du type de l'objet « y », c'est à dire qu'il n'y a plus de « types construits » à reconstituer, l'agent de sérialisation conclut que l'objet (34) résultat est complètement créé.

Dans une variante, la création ou allocation d'un nouvel objet peut se faire à n'importe quel moment du processus de désérialisation, à partir de la lecture de la balise NEW indiquant ce nouvel objet, jusqu'au début du remplissage de cet objet.

Dans une variante, l'index utilisé lors de la désérialisation d'un objet est identique à l'identifiant de l'objet dans la carte.

Au sein de la carte, les objets sont sérialisés vers le flux (42) de sortie depuis la file de sortie « qout », qui est directement accessible à l'instruction « SendObject() » fournie par la classe « ObjectIOApplet » et utilisée par l'agent ou l'application destinataire.

Dans l'implémentation du procédé selon l'invention, lorsque l'hôte veut déclencher dans la carte une opération de sérialisation, il utilise une instruction, par exemple selon la syntaxe Java® « card.Serialize Out ».

Dans la figure 5 est illustrée la sérialisation d'un objet (41) logiciel structuré, ou objet de départ, à travers l'analyse des composants de la structure de ce même objet (41) structuré puis le codage sous forme de données mémorisées dans le flux (42) de sortie correspondant à cet objet (41) de départ.

Cette fonctionnalité de sérialisation est implémentée dans une méthode, c'est à dire une action ou procédure disponible pour un objet d'un type déterminé, et peut présenter la syntaxe Java® suivante :

```
Typeobject:: (Object, OutputStream)
```

étant une méthode utilisée pour le codage d'un objet « Object » de type « Typeobject » vers le flux de sortie « OutputStream ».

```
Typeobject::getSuper()
```

étant une méthode utilisée pour obtenir les informations concernant le type et les types construits d'un objet, lors du codage d'un l'objet de type « Typeobject ».

Les types de bases sont en général des types prévus et gérés par l'environnement ou le langage de programmation, par opposition à des types,

dits types construits, définis comme une combinaison de plusieurs objets. Des types de base courants sont par exemple les types entier, booléen, octets (« integer », « boolean », « byte ») pour un environnement embarqué, et également entier long, réel, réel long, caractère (« long », « real », « double », « char ») pour des environnements plus complets.

Pour cette sérialisation, l'agent (212) de sérialisation parcourt de façon récursive l'ensemble de la structure, ou du graphe, de l'objet (42) de départ à sérialiser, et en analyse les éléments (410, 412, 413, 414, 411), en commençant par l'objet ou élément « racine », « x » dans l'exemple. Le graphe de l'objet (42) de départ présenté dans l'exemple est le même que décrit ci-dessus pour la figure 4. Pour chacun des éléments du graphe présentant un type construit, l'agent de sérialisation fait appel à un agent (TM0, TM1, TM2) correspondant à ce même type construit.

La description dans le flux (42) de sortie de l'objet (41) de départ commence par l'écriture d'une balise NEW suivie de l'identifiant du type de l'élément racine, l'objet « x » de type 2 dans l'exemple. Cet objet racine est alors désigné comme objet (OBJ) en cours. Par ailleurs, cet objet reçoit un index, 0 dans l'exemple, puis son type et son index sont introduits (action « push ») dans la pile (TYST) de types.

La description se poursuit ensuite par l'écriture d'une balise DATA, suivie des données indiquant les valeurs ou références correspondant au contenu de cet objet racine. Du fait que l'objet racine « x » contient un objet « y » (414), qui est d'un type construit, la description de l'objet racine comprendra une balise NEW suivie du type de cet objet « y », « NEW 0 » dans l'exemple, en lieu et place de la valeur de cet objet « y ». Lors de l'écriture de cette balise NEW, un index lui est attribué, 1 dans l'exemple. Puis l'index et le type de ce nouvel objet sont introduits (action « push ») dans la pile (TYST) de types.

dits types construits, définis comme une combinaison de plusieurs objets. Des types de base courants sont par exemple les types entier, booléen, octets (« integer », « boolean », « byte ») pour un environnement embarqué, et également entier long, réel, réel long, caractère (« long », « real », « double », « char ») pour des environnements plus complets.

Pour cette sérialisation, l'agent (212) de sérialisation parcourt de façon récursive l'ensemble de la structure, ou du graphe, de l'objet (41) de départ à sérialiser, et en analyse les éléments (410, 412, 413, 414, 411), en commençant par l'objet ou élément « racine », « x » dans l'exemple. Le graphe de l'objet (41) de départ présenté dans l'exemple est le même que décrit ci-dessus pour la figure 4. Pour chacun des éléments du graphe présentant un type construit, l'agent de sérialisation fait appel à un agent (TM0, TM1, TM2) correspondant à ce même type construit.

La description dans le flux (42) de sortie de l'objet (41) de départ commence par l'écriture d'une balise NEW suivie de l'identifiant du type de l'élément racine, l'objet « x » de type 2 dans l'exemple. Cet objet racine est alors désigné comme objet (OBJ) en cours. Par ailleurs, cet objet reçoit un index, 0 dans l'exemple, puis son type et son index sont introduits (action « push ») dans la pile (TYST) de types.

La description se poursuit ensuite par l'écriture d'une balise DATA, suivie des données indiquant les valeurs ou références correspondant au contenu de cet objet racine. Du fait que l'objet racine « x » contient un objet « y » (414), qui est d'un type construit, la description de l'objet racine comprendra une balise NEW suivie du type de cet objet « y », « NEW 0 » dans l'exemple, en lieu et place de la valeur de cet objet « y ». Lors de l'écriture de cette balise NEW, un index (4142) lui est attribué, 1 dans l'exemple. Puis l'index et le type de ce nouvel objet sont introduits (action « push ») dans la pile (TYST) de types.

Une fois terminée la description du contenu de l'objet (OBJ), c'est à dire l 'objet « x » (410), son index est mémorisé dans la pile (OBJST) d'objets

Une fois terminée la description du contenu de l'objet (OBJ), c'est à dire l 'objet « x » (410), son index est mémorisé dans la pile (OBJST) d'objets avec l'identifiant de cet objet. L'agent de sérialisation consulte alors la piles de types, en extrait (action « pop ») le type et l'index de l'objet « y ». Il mémorise cet objet « y » comme nouvel objet (OBJ) en cours, et commence alors la description du contenu de cet objet « y ». Cet objet étant constitué d'une simple référence à l'objet racine « x », les données écrites dans le flux de sortie seront constituées d'une balise REF suivie d'une donnée valant l'index de l'objet à désigner en référence, c'est à dire l'objet « x » d'index 0 dans l'exemple.

Une fois terminée la description du contenu de cet objet en cours, c'est à dire l'objet « y » (414), son index est mémorisé dans la pile (OBJST) d'objets avec l'identifiant de cet objet. L'agent de sérialisation consulte alors la piles de types, en extrait (action « pop ») le type de l'objet « x » qui étaient mémorisés sous l'objet « y » dans cette pile de types. Du fait que l'index de l'objet « x » est déjà mémorisé dans la pile (OBJST) d'objets, l'agent de sérialisation sait que cet objet « x » a déjà été sérialisé, ou décrit. Il consulte donc à nouveau la pile de types, constate qu'elle est vide, et conclue donc que l'ensemble des éléments composants de l'objet (34) de départ ont été entièrement décrits dans le flux (42) de sortie.

Du fait de la récursivité de ces algorithmes, il est ainsi possible de réaliser ces opérations de sérialisation et désérialisation à travers un code programme occupant suffisamment peu de place en mémoire pour pouvoir être mémorisé dans une plate-forme embarquée, par exemple une carte à puce ou un objet informatisé portable au standard JavaCard. La concision de ces algorithmes permet également d'exécuter ces opérations dans un processeur de faible puissance, comme ceux dont disposent de telles plates-formes embarquées.

De façon à équilibrer les quantités de données stockées sous forme intermédiaire, les différentes opérations de lecture, écriture, sérialisation et désérialisation demandées par l'application (11) hôte peuvent être entrelacées

avec l'identifiant de cet objet. L'agent de sérialisation consulte alors la piles de types, en extrait (action « pop ») le type et l'index de l'objet « y ». Il mémorise cet objet « y » comme nouvel objet (OBJ) en cours, et commence alors la description du contenu de cet objet « y ». Cet objet étant constitué d'une simple référence (4141) à l'objet racine « x », les données écrites dans le flux de sortie seront constituées d'une balise REF suivie d'une donnée valant l'index de l'objet à désigner en référence, c'est à dire l'objet « x » d'index 0 dans l'exemple.

Une fois terminée la description du contenu de cet objet en cours, c'est à dire l'objet « y » (414), son index est mémorisé dans la pile (OBJST) d'objets avec l'identifiant de cet objet. L'agent de sérialisation consulte alors la piles de types, en extrait (action « pop ») le type de l'objet « x » qui étaient mémorisés sous l'objet « y » dans cette pile de types. Du fait que l'index de l'objet « x » est déjà mémorisé dans la pile (OBJST) d'objets, l'agent de sérialisation sait que cet objet « x » a déjà été sérialisé, ou décrit. Il consulte donc à nouveau la pile de types, constate qu'elle est vide, et conclue donc que l'ensemble des éléments composants de l'objet (41) de départ ont été entièrement décrits dans le flux (42) de sortie.

Du fait de la récursivité de ces algorithmes, il est ainsi possible de réaliser ces opérations de sérialisation et désérialisation à travers un code programme occupant suffisamment peu de place en mémoire pour pouvoir être mémorisé dans une plate-forme embarquée, par exemple une carte à puce ou un objet informatisé portable au standard JavaCard. La concision de ces algorithmes permet également d'exécuter ces opérations dans un processeur de faible puissance, comme ceux dont disposent de telles plates-formes embarquées.

De façon à équilibrer les quantités de données stockées sous forme intermédiaire, les différentes opérations de lecture, écriture, sérialisation et désérialisation demandées par l'application (11) hôte peuvent être entrelacées

dans une ou plusieurs boucles de programme. Une telle boucle exécute à chaque itération une commande de chacune de ces opérations, par exemple selon la syntaxe Java® suivante :

```
Do
    Do card.Serialize out While (ok out)
    Do card.Read While (data read)
    While (data in) card.Write
    While (ok in) card.Serialize in
Loop
```

Dans cet exemple, les première et dernière lignes (« Do » et « Loop ») détermine la répétition d'une boucle constituée des quatre lignes de code qu'elles encadrent. Une telle boucle peut bien sûr être combinée avec d'autres opérations, et comporter diverses conditions d'interruption de la répétition.

A l'intérieur de la boucle, la première ligne indique de déclencher dans la carte une session de sérialisation, vers le flux de sortie de la carte, des objets précédemment mémorisés dans la file de sortie « qout » par une commande « SendObject() ». Cette session est ensuite répétée tant qu'une condition dénommée « ok out » est remplie, par exemple tant qu'il y a des objets à envoyer dans la file « qout » et que le flux de sortie n'est pas plein.

La deuxième ligne indique de déclencher une session de lecture par l'hôte des données contenues dans le flux de sortie de la carte. Cette session est ensuite répétée tant qu'une condition dénommée « data read » est remplie, par exemple tant que des données sont reçues depuis la carte.

La troisième ligne indique d'effectuer et répéter une session d'écriture dans le flux d'entrée de données venant de l'hôte. Cette session ne s'effectue et ne se répète que si et tant qu'une condition dénommée « data in » est remplie. Par exemple, tant qu'il y a des données à envoyer à la carte, et que le flux d'entrée de la carte n'est pas plein.

La quatrième ligne indique d'effectuer et répéter une session de désérialisation des données contenues dans le flux d'entrée de la carte, vers la file d'entrée « qin » d'objets structurés, d'où ils sont extraits par une commande « process(Object) ». Cette session ne s'effectue et ne se répète que si et tant

qu'une condition dénommée « ok in » est remplie. Par exemple, tant qu'il y a des données à désérialiser dans le flux d'entrée de la carte.

Dans le cas d'une carte passive, par exemple au standard JavaCard®, c'est typiquement la fin de l'opération de sérialisation d'un objet qui déclenchera son traitement, par exemple en appelant automatiquement la méthode « process(Object) ».

Il est à noter que seules les opérations à effectuer du côté de la carte sont illustrées dans cet exemple. Les opérations symétriques effectuées du côté de l'hôte peuvent être réalisées de façon similaire comme de façon totalement différente, selon les ressources matérielles et logicielles disponibles, sans sortir de l'esprit de l'invention.

Du fait que ces différentes opérations complémentaires sont entrelacées dans une boucle logicielle pouvant se répéter de façon récursive, les différentes phases constituant le transfert d'un objet structuré peuvent être réalisées en parallèle, dans le cadre d'une seule exécution ou d'une seule commande de transfert. En utilisant des flux de données mémorisés de façon circulaire réutilisant indéfiniment un même espace mémoire comme décrit plus haut, une même commande peut déclencher le transfert complet d'un objet structuré sans limite de taille malgré les capacités limitées de la plate-forme embarquée.

Lorsque l'ensemble des opérations de conversions décrites ci-dessus est implémenté, c'est à dire programmé, dans une ou plusieurs procédures chargées dans l'hôte et dans la plate-forme embarquée. Ces procédures peuvent alors être accessibles à l'utilisateur à travers quelques commandes simples.

Dans le mode de réalisation décrit ici, appliqué au langage Java® et à l'environnement JavaCard®, le programmeur d'une application n'a ainsi qu'à utiliser ces quelques commandes simples pour réaliser son application. Ces commandes réalisent l'ensemble des opérations intermédiaires de façon

transparente pour l'utilisateur, c'est à dire sans qu'il ait besoin de se préoccuper du fonctionnement interne de leur mécanisme.

A titre d'exemple, la classe « ObjectIOProxy », chargée dans la station de traitement ou le terminal hôte, fournit ainsi les commandes « WriteObject() » et « ReadObject() ».

De même, la classe « ObjectIOApplet », chargée dans la plate-forme embarquée, fournit les commandes « Process(Object) » et « SendObject() ». De façon typique, la classe « ObjectIOApplet » implémente ces commandes par une technique connue de d'extension, c'est à dire par héritage de la classe « IOApplet » qui contient la méthode « process(APDU) ». C'est à dire en ajoutant à cette méthode du code programme qui vient s'ajouter à la méthode initiale et modifier ou remplacer son fonctionnement tel que prévu dans la méthode initiale.

L'instruction « WriteObject() » est utilisée par l'utilisateur dans son application hôte pour réaliser l'envoi d'un objet structuré à l'application de la carte et y déclencher un traitement.

La méthode « Process(Object) » est appelée automatiquement par la carte à la fin de la réception et dé-sérialisation d'un objet structuré. Le contenu de cette méthode est alors programmé par l'utilisateur dans la partie de son application chargée dans la plate-forme embarquée pour réaliser les opérations souhaitées à partir de cet objet. Cette méthode est alors utilisée de façon similaire à la commande standard « process(APDU) » existant en JavaCard® pour les seules données au format APDU.

L'instruction « SendObject() » est utilisée par l'utilisateur dans la partie de son application chargée dans la plate-forme embarquée, par exemple à l'intérieur du code de la méthode « Process(Object) », pour préparer l'envoi d'un ou plusieurs objets structurés de la plate-forme embarquée vers l'hôte. Cette instruction est alors utilisée de façon similaire à la commande standard « sendAPDU() » existant en JavaCard® pour les seules données au format APDU.

L'instruction « ReadObject() » est utilisée par l'utilisateur dans son application hôte pour réaliser la réception depuis la plate-forme embarquée du ou des objets structurés que l'application de la carte à préparé dans ce but.

On comprend bien qu'ainsi il est plus aisé à un programmeur, réalisant une application comportant une telle plate-forme embarquée, de transmettre des objets logiciels structurés entre cette plate-forme embarquée et un hôte ou terminal, même si les moyens de communication de cette plate-forme embarquée ne sont aptes qu'à transmettre des données sous forme d'octets ou d'entiers.

Pour le programmeur d'une application embarquée, il peut être utile de disposer de quelques commandes simples permettant de réaliser la désallocation d'un objet logiciel structuré ou la remise à zéro ou effacement de l'espace mémoire correspondant, ainsi que la duplication d'un objet logiciel structuré. Cela est particulièrement vrai lorsque l'environnement de programmation de la plate-forme embarquée ne comprend pas d'outil logiciel (« garbage collector » en anglais) gérant la réutilisation de l'espace mémoire déjà alloué, par exemple dans l'environnement JavaCard®.

Dans un mode de réalisation du procédé selon l'invention, l'étape d'analyse de chacun des composants d'un objet logiciel structuré au cours de sa sérialisation peut effectuer selon différentes options. Plusieurs de ces options peuvent être prévues dans l'implémentation d'une même commande de sérialisation, chaque exécution de cette commande étant assortie d'un paramètre indiquant quelle option doit être utilisée. Ces différentes options peuvent également être utilisées séparément dans différentes implémentations de la commande de sérialisation.

Dans une de ces options, lors de l'analyse de chacun des composants d'un objet logiciel structuré au cours de sa sérialisation, l'agent (212) de sérialisation effectue une désallocation de l'espace mémoire contenant ce composant, ou marque ce même composant comme pouvant être réutilisé pour

une nouvelle allocation. Ainsi, une sérialisation selon cette option, dite sérialisation destructive, effectue une conversion d'un objet structuré suivie d'une libération de son espace mémoire.

Dans un mode de réalisation, le procédé selon l'invention peut prévoir une option ou une commande réalisant une telle sérialisation destructive d'un objet structuré mémorisé dans la carte, hors de toute transmission, par exemple dans l'implémentation de la classe « ObjectIOApplet ». On comprend bien qu'ainsi le procédé selon l'invention permet à l'utilisateur de réaliser aisément la réutilisation de l'espace mémoire occupé par tout ou partie de certains des objets logiciels de son application.

Dans un mode de réalisation (non représenté), le procédé selon l'invention prévoit une option ou une commande réalisant la désérialisation d'un objet structuré en ré-utilisant l'espace mémoire dans la carte d'un objet structuré précédemment détruit, comme décrit ci-dessus. Cette désérialisation est effectuée hors de toute transmission, à partir d'un flux de données contenant des données toutes identiques ou sans signification. Après allocation et écriture d'un tel objet, l'espace mémoire réutilisé ne contiendra plus aucune donnée provenant de l'objet qui l'occupait précédemment. Lorsque cette opération est implémentée par exemple dans la classe « ObjectIOApplet », on comprend bien qu'ainsi le procédé selon l'invention permet à l'utilisateur de programmer aisément l'effacement complet d'un espace mémoire correspondant à une allocation déterminée.

Dans un mode de réalisation (non représenté), le procédé selon l'invention prévoit une option ou une commande réalisant la sérialisation d'un objet structuré de départ vers une succession linéaire de données, hors de toute transmission. Cette même succession linéaire de données est alors désérialisée en un objet résultat identique à l'objet de départ mais utilisant un autre espace mémoire. Lorsque cette opération est implémentée par exemple dans la classe « ObjectIOApplet », on comprend bien qu'ainsi le procédé selon

l'invention permet à l'utilisateur de programmer aisément la duplication à l'identique d'un objet logiciel structuré.

Il doit être évident pour les personnes versées dans l'art que la présente invention permet des modes de réalisation sous de nombreuses autres formes spécifiques sans l'éloigner du domaine d'application de l'invention comme revendiqué. Par conséquent, les présents modes de réalisation doivent être considérés à titre d'illustration, mais peuvent être modifiés dans le domaine défini par la portée des revendications jointes, et l'invention ne doit pas être limitée aux détails donnés ci-dessus.

REVENDICATIONS

1. Procédé de conversion de données utilisable par une station (2) informatique, dite plate-forme embarquée, comprenant un objet portable incluant au moins un processeur, des moyens de mémorisation, et des moyens de communication aptes à échanger des informations avec un terminal sous la forme d'une ou plusieurs successions linéaires de données, caractérisé en ce qu'il comporte une étape de conversion d'un ensemble de données, dans un sens ou dans l'autre, entre d'une part un agencement en une succession linéaire de données et d'autre part un agencement structuré décrivant ou représentant un ou plusieurs objets logiciels structurés ou hiérarchisés suivant les critères d'un langage de programmation orienté objet.

2. Procédé selon la revendication précédente, caractérisé en ce qu'il comporte des étapes de :

- conversion, ou sérialisation, d'un premier ensemble de données à transmettre comportant ou représentant un ou plusieurs objets (31, 41) logiciels structurés ou hiérarchisés suivant les critères d'un langage de programmation orienté objet, depuis un agencement structuré décrivant ou représentant cet objet vers une succession linéaire de données représentant ce premier ensemble de données ;

- transmission de cette succession linéaire de données par des moyens (200, respectivement 100) de communication, depuis la plate-forme (2) embarquée vers au moins un hôte (1), c'est à dire un terminal ou une station informatique reliée au terminal, ou de cet hôte (1) vers la plate-forme (2) embarquée ;

- conversion après transmission, ou dé-sérialisation, de cette succession linéaire de données vers un ensemble de données agencé en un ou plusieurs objets (34, respectivement 44) logiciels structurés reproduisant ou représentant le premier ensemble de données.

3. Procédé selon l'une des revendications précédentes, caractérisé en ce que le terminal hôte (1) envoie des informations à la plate-forme (2) embarquée en utilisant un agent (101) logiciel, dit fonction de transmission, ces informations étant reçues dans la plate-forme embarquée par une fonction (201) de réponse apte à déclencher un traitement (2210) de ces données par au moins un agent (221) logiciel destinataire mémorisé dans la plate-forme embarquée et faisant partie d'au moins une application (21), le procédé comprenant des étapes de :

- réception, par un agent (211) de communication en lieu et place de l'agent (221) destinataire, d'un ensemble (32) de données reçues par la fonction (201) de réponse à l'intention de cet agent logiciel destinataire, cet ensemble de données étant agencé en une succession linéaire de données ;

- conversion de cet ensemble de données en au moins un objet (34) logiciel, structuré ou hiérarchisé suivant les critères d'un langage de programmation orienté objet ;

- transmission de cet objet (34) logiciel structuré à l'agent (221) destinataire et déclenchement d'un traitement (2210) en fonction de cet objet par cet agent destinataire.

4. Procédé selon l'une des revendications précédentes, caractérisé en ce qu'il comporte des étapes de :

- réception par la fonction (201) de réponse, en provenance de la fonction (101) de transmission de l'hôte (1), d'au moins une donnée sous forme d'au moins un paramètre (32) d'envoi, et transmission de ce paramètre à un agent (211) de communication mémorisé ou exécuté dans la plate-forme (2) embarquée ;

- conversion, ou concaténation, par l'agent (211) de communication d'au moins un paramètre (32) d'envoi, transmis par l'agent (201) de réponse, en

un ensemble de données agencé en une succession linéaire de données et mémorisation de ces données dans un flux (33) d'entrée dans la plate-forme (2) embarquée ;

- conversion, ou dé-sérialisation, par un agent (212) de sérialisation, mémorisé ou exécuté dans la plate-forme (2) embarquée, d'au moins une partie des données mémorisées dans ce flux (33) d'entrée vers un ensemble de données comprenant ou représentant au moins un objet (34) logiciel structuré ;

- réception de cet objet (34) logiciel structuré ou de ses références par l'agent (221) destinataire.

5. Procédé selon l'une des revendications précédentes, caractérisé en ce qu'il comporte des étapes de :

- transmission d'un objet (41) logiciel structuré ou de sa représentation depuis un agent (221) logiciel faisant partie d'une application (22) exécutée ou mémorisée dans une plate-forme (2) embarquée vers un agent (212) de sérialisation exécuté ou mémorisé dans cette plate-forme embarquée ;

- conversion, ou sérialisation, par cet agent (212) de sérialisation de cet objet (41) logiciel structuré vers un ensemble de données agencé en une succession linéaire de données et mémorisation de ces données dans un flux (42) de sortie dans la plate-forme embarquée ;

- conversion par un agent (211) de communication, mémorisé ou exécuté dans la plate-forme (2) embarquée, d'au moins une partie des données mémorisées dans ce flux (42) de sortie vers un ensemble de paramètres (43) de réponse aptes à être transmis par la fonction (201) de réponse ;

- transmission de ces paramètres (43) de réponse depuis la plate-forme (2) embarquée vers le terminal hôte (1) par la fonction (201) de réponse, de sa

propre initiative ou en réponse à la fonction (101) de transmission du terminal hôte (1).

6. Procédé selon l'une des revendications précédentes, caractérisé en ce que la succession linéaire de données mémorisée dans le flux (33) d'entrée ou le flux (42) de sortie représente un ou plusieurs objets (31, respectivement 41) logiciels structurés ou hiérarchisés en utilisant une ou plusieurs données, dites balises, d'une ou plusieurs valeurs déterminées représentant chacune une action déterminée à effectuer lors de la désérialisation de cette succession linéaire de données.

7. Procédé selon l'une des revendications précédentes, caractérisé en ce qu'au moins une balise est définie comme représentant une des actions suivantes :

- ajout d'un nouvel élément à la structure de l'objet structuré représenté par la succession linéaire de données ;
- référence à un élément ou objet, dit objet source, en tant que source de la valeur de tout ou partie d'un élément composant l'objet structuré ;
- indication que la ou les données suivantes représentent le contenu d'un élément composant l'objet structuré ;
- indication d'une absence de contenu d'un élément composant l'objet structuré.

8. Procédé selon l'une des revendications précédentes, caractérisé en ce que l'agent (212) de sérialisation effectue la sérialisation d'un objet (41) structuré, dit objet de départ, en un ensemble (42) linéaire de données suivant un procédé, dit procédé de sérialisation, traitant au moins un des objets (410 à 414), dits éléments, composant la structure ou l'arborescence de cet objet (41) structuré de départ, selon des étapes de :

- détection, par l'agent (212) de sérialisation du type (4100) d'un élément (410, 414), dit objet en cours, composant la structure ou l'arborescence de cet objet (41) structuré ;

- mémorisation dans le flux (42) de sortie d'une donnée représentant une balise indiquant l'ajout d'un nouvel élément, suivie d'une donnée représentant le type (TYOBJ) de l'objet en cours ;

- mémorisation dans le flux de sortie, à la suite des éléments qui y sont déjà présents, par un agent de sérialisation de type (TM0, TM1, TM2) associé au type (TYOBJ) de l'objet en cours,

  - soit d'au moins une donnée (424, 425, 429) représentant la valeur de tout ou partie (412, 413, 411) de l'objet (41) structuré ;

  - soit d'au moins une donnée (4211, 4212) représentant une balise (4211) indiquant une référence à un objet (410) en tant que source de la valeur de tout ou partie (414) de l'objet (41) structuré, cette balise étant suivie d'une donnée (4212) identifiant ledit objet (410) source ;

9. Procédé selon l'une des revendications précédentes, caractérisé en ce que le procédé de sérialisation effectue la conversion d'un objet (41) structuré vers le flux (42) de sortie en mémorisant au fur et à mesure de ses itérations le type de chaque objet en cours dans une pile (TYST) mémoire, dite pile de types, dont les emplacements sont lus dans l'ordre inverse de leur ordre de mémorisation.

10. Procédé selon l'une des revendications précédentes, caractérisé en ce que l'agent (212) de sérialisation effectue la désérialisation d'un ensemble linéaire de données en au moins un objet (34) structuré résultat, suivant un procédé, dit procédé de désérialisation, traitant chacune des données mémorisées dans le flux (33) d'entrée, selon des étapes de :

- détection, par l'agent (212) de sérialisation du type (4100) d'un élément (410, 414), dit objet en cours, composant la structure ou l'arborescence de cet objet (41) structuré ;

- mémorisation dans le flux (42) de sortie d'une donnée représentant une balise indiquant l'ajout d'un nouvel élément, suivie d'une donnée représentant le type (TYOBJ) de l'objet en cours ;

- mémorisation dans le flux de sortie, à la suite des éléments qui y sont déjà présents, par un agent de sérialisation de type (TM0, TM1, TM2) associé au type (TYOBJ) de l'objet en cours,

  - soit d'au moins une donnée représentant la valeur de tout ou partie (412, 413, 411) de l'objet (41) structuré ;

  - soit d'au moins une donnée représentant une balise indiquant une référence à un objet (410) en tant que source de la valeur de tout ou partie (414) de l'objet (41) structuré, cette balise étant suivie d'une donnée identifiant ledit objet (410) source ;

9. Procédé selon l'une des revendications précédentes, caractérisé en ce que le procédé de sérialisation effectue la conversion d'un objet (41) structuré vers le flux (42) de sortie en mémorisant au fur et à mesure de ses itérations le type de chaque objet en cours dans une pile (TYST) mémoire, dite pile de types, dont les emplacements sont lus dans l'ordre inverse de leur ordre de mémorisation.

10.Procédé selon l'une des revendications précédentes, caractérisé en ce que l'agent (212) de sérialisation effectue la désérialisation d'un ensemble linéaire de données en au moins un objet (34) structuré résultat, suivant un procédé, dit procédé de désérialisation, traitant chacune des données mémorisées dans le flux (33) d'entrée, selon des étapes de :

détection, par l'agent (212) de sérialisation du type (4100) d'un élément (410, 414), dit objet en cours, composant la structure ou l'arborescence de cet objet (41) structuré ;

- mémorisation dans le flux (42) de sortie d'une donnée représentant une balise indiquant l'ajout d'un nouvel élément, suivie d'une donnée représentant le type (TYOBJ) de l'objet en cours ;

- mémorisation dans le flux de sortie, à la suite des éléments qui y sont déjà présents, par un agent de sérialisation de type (TM0, TM1, TM2) associé au type (TYOBJ) de l'objet en cours,

  - soit d'au moins une donnée représentant la valeur de tout ou partie (412, 413, 411) de l'objet (41) structuré ;

  - soit d'au moins une donnée représentant une balise indiquant une référence à un objet (410) en tant que source de la valeur de tout ou partie (414) de l'objet (41) structuré, cette balise étant suivie d'une donnée identifiant ledit objet (410) source ;

9. Procédé selon l'une des revendications précédentes, caractérisé en ce que le procédé de sérialisation effectue la conversion d'un objet (41) structuré vers le flux (42) de sortie en mémorisant au fur et à mesure de ses itérations le type de chaque objet en cours dans une pile (TYST) mémoire, dite pile de types, dont les emplacements sont lus dans l'ordre inverse de leur ordre de mémorisation.

10.Procédé selon l'une des revendications précédentes, caractérisé en ce que l'agent (212) de sérialisation effectue la désérialisation d'un ensemble linéaire de données en au moins un objet (34) structuré résultat, suivant un procédé, dit procédé de désérialisation, traitant chacune des données mémorisées dans le flux (33) d'entrée, selon des étapes de :

- lecture par l'agent de sérialisation d'au moins une donnée mémorisée dans le flux d'entrée à la suite des données précédemment traitées ;

- analyse de cette donnée et réalisation d'une action correspondant à cette donnée.

11. Procédé selon l'une des revendications précédentes, caractérisé en ce que le procédé de désérialisation comprend le remplissage d'un élément, dit objet en cours, c'est à dire l'affectation d'une valeur directe ou indirecte à tout ou partie de cet objet en cours, cet élément composant tout ou partie de la structure de l'objet (34) structuré résultat, la fin du remplissage de l'objet en cours déclenchant

- la lecture d'une donnée (TYT2) représentant un type d'objet mémorisé dans le prochain emplacement d'une structure de mémoire, dite pile (TYST) de types, et l'effacement de cette donnée de cet emplacement ;

- la mémorisation, comme type (TYOBJ) d'un nouvel objet en cours, du type représenté par la donnée (TYT2) lue dans la pile de types.

12. Procédé selon l'une des revendications précédentes, caractérisé en ce que l'objet (34) structuré créé par le procédé de désérialisation est considéré comme complet et transmis à l'agent (221) logiciel ou l'application (22) qui en est destinataire lorsque la pile (TYST) de types est vide.

13. Procédé selon l'une des revendications précédentes, caractérisé en ce que l'action de désérialisation correspondant à une donnée (321, 326) représentant une balise, dite balise « NEW », indiquant un nouvel élément comprend des étapes de :

- lecture d'au moins une donnée (322, 327) suivante dans le flux d'entrée ;

- mémorisation du type d'objet, représenté par cette même donnée (322, 327) suivante, dans une pile mémoire, dite pile (TYST) de types, à la suite des types qui y sont déjà éventuellement mémorisés ;

14. Procédé selon l'une des revendications précédentes, caractérisé en ce que la pile (TYDT) de types utilisée par le procédé de désérialisation comprend une pile de type LIFO, c'est à dire dont les emplacements sont lus dans l'ordre inverse de leur ordre de mémorisation.

15. Procédé selon l'une des revendications précédentes, caractérisé en ce que le procédé de désérialisation comprend une étape de création d'un élément (410 à 414) composant la structure de l'objet (34) logiciel structuré résultat, par allocation d'un nouvel espace mémoire ou par réutilisation d'une allocation libre, cette création étant réalisée par un agent (TM0, TM1, TM2) de contrôle de type correspondant au type de l'élément à créer.

16. Procédé selon l'une des revendications précédentes, caractérisé en ce que l'étape de création d'un élément (410, 414) composant la structure de l'objet (34) logiciel structuré résultat a lieu entre l'étape de lecture d'une donnée (322, 327) indiquant la création de cet élément et la première étape de remplissage de ce même élément.

17. Procédé selon l'une des revendications précédentes, caractérisé en ce que l'action correspondant à une donnée (324, 325, 329), dite donnée simple, c'est à dire ne représentant pas une balise, comprend une étape de mémorisation de la valeur de cette donnée dans l'espace mémoire alloué à l'objet en cours.

18. Procédé selon l'une des revendications précédentes, caractérisé en ce qu'au cours du procédé de désérialisation, un index (4102) d'objet est attribué à au moins un élément (410) composant la structure de l'objet (34) structuré résultat, cet index d'objet identifiant cet élément de façon unique et lui permettant d'être désigné ou référencé par d'autres objets ou éléments (414).

19. Procédé selon l'une des revendications précédentes, caractérisé en ce que l'action correspondant à une donnée (3211) représentant une balise, dite balise « REF », indiquant une référence comprend des étapes de :

- lecture d'au moins une donnée (3212) suivante dans le flux (32) d'entrée ;
- mémorisation dans l'espace (3441) mémoire alloué à l'objet (344) en cours, à la suite des données déjà mémorisées, d'une donnée désignant un objet (340) comme source de la valeur de tout ou partie de l'objet en cours, cet objet ou élément (340) source étant identifié par ladite donnée (3212) suivante.

20. Procédé selon l'une des revendications précédentes, caractérisé en ce que le remplissage de l'objet en cours est considéré comme terminé lorsque les données mémorisées dans l'espace mémoire qui lui est alloué correspondent à une longueur déterminée, cette longueur étant lue en mémoire dans la carte (2) par un agent (TM0, TM1, TM2) de contrôle de type ou par un agent (TMG) gestionnaire de types mémorisé dans la plate-forme embarquée.

21. Procédé selon l'une des revendications précédentes, caractérisé en ce que l'envoi par la plate-forme embarquée (2), vers l'hôte (1), d'une succession (42) linéaire de données d'une longueur déterminée, s'effectue selon un procédé itératif, dit de lecture de données, comprenant des étapes récursives de :

- envoi par l'hôte d'une commande de communication avec passage d'au moins un paramètre d'envoi représentant la longueur des données déjà reçues ;
- réception par la plate-forme embarquée du paramètre d'envoi et comparaison avec la succession linéaire de données à transmettre ;

- réponse à la commande de communication par l'envoi d'au moins un paramètre (43) de retour représentant la ou les données suivant immédiatement les données déjà reçues par l'hôte ;

- réception par l'hôte du paramètre (43) de retour de la commande de communication et mémorisation des données qu'il représente à la suite des données déjà reçues.

22. Procédé selon l'une des revendications précédentes, caractérisé en ce que la réception par la plate-forme embarquée (2), en provenance de l'hôte (1), d'une succession (43) linéaire de données d'une longueur déterminée, s'effectue selon un procédé itératif, dit d'écriture de données, comprenant des étapes récursives de :

- envoi par l'hôte d'une commande de communication avec passage d'au moins un premier paramètre (32) d'envoi représentant la ou les données suivant immédiatement la partie déjà transmise de la succession linéaire de données à transmettre, ainsi éventuellement qu'un deuxième paramètre d'envoi représentant la position, dans la succession à transmettre, des données représentées par le premier paramètre d'envoi ou la longueur des données déjà transmises ;

- réception par la plate-forme embarquée du ou des paramètres (32) d'envoi de la commande de communication ;

- mémorisation dans un flux (33) d'entrée dans la plate-forme embarquée des données représentées par le premier paramètre (32) d'envoi à la suite des données déjà reçues.

23. Procédé selon l'une des revendications précédentes, caractérisé en ce que le procédé d'écriture de données comprend en outre les étapes suivantes :

- comparaison par la plate-forme embarquée du deuxième paramètre d'envoi et comparaison avec la longueur des données déjà reçues ;

- retour par la plate-forme embarquée d'au moins un paramètre de réponse représentant soit la longueur des données déjà reçues soit une donnée représentant le résultat de cette comparaison soit les deux ;

24. Procédé selon l'une des revendications précédentes, caractérisé en ce qu'au moins deux des procédés de sérialisation, désérialisation, lecture de données, ou écriture de données sont effectués en parallèle ou de façon entrelacée par répétition d'une étape comprenant l'exécution successive d'au moins une étape de chacun de ces procédés.

25. Procédé selon l'une des revendications précédentes, caractérisé en ce que le flux d'entrée ou le flux de sortie ou les deux sont mémorisés sous la forme de structures circulaires de mémoire, ces deux flux pouvant partager la même structure circulaire.

26. Procédé selon l'une des revendications précédentes, caractérisé en ce que la plate-forme embarquée comporte un environnement programmable apte à mémoriser et exécuter au moins une application réalisée par un programmeur, la fonction de communication étant compatible avec le format « APDU » défini dans la norme ISO 7816.

27. Procédé de conversion selon la revendication précédente, cecq des opérations de désérialisation puis traitement d'un objet reçu sont déclenchées par la réception d'au moins une commande au format APDU comprenant au moins une donnée indiquant la réception d'un objet structuré.

28. Procédé selon l'une des revendications précédentes, caractérisé en ce que la plate-forme embarquée comporte un environnement programmable compatible avec le standard JavaCard (marque déposée).

29. Procédé selon l'une des revendications précédentes, caractérisé en ce qu'au moins une des applications exécutées sur l'hôte ou la plate-forme embarquée est programmée en langage Java®.

30. Procédé selon l'une des revendications précédentes, caractérisé en ce que les procédés de lecture de données, d'écriture de données, de dé-sérialisation ou de sérialisation sont mis en œuvre à travers leur implémentation dans au moins une classe mémorisée dans l'hôte ou dans la plate-forme embarquée, cette implémentation comprenant au moins l'une des commandes suivantes :

- une commande d'écriture d'objet, effectuant la transmission d'un objet (31) structuré à au moins un agent (221) de la plate-forme embarquée, par utilisation du procédé de sérialisation de cet objet, puis du procédé d'écriture de données, puis du procédé de dé-sérialisation ;

- une commande de lecture d'objet, effectuant la lecture d'un objet (41) structuré depuis au moins un agent (221) de la plate-forme embarquée, par utilisation du procédé de sérialisation puis du procédé de lecture de données, puis du procédé de désérialisation ;

- une commande de désallocation d'un objet structuré mémorisé dans la plate-forme embarquée, par utilisation du procédé de sérialisation, celui-ci comprenant en outre une étape de mémorisation de la libération ou désallocation de l'espace mémoire alloué à chaque composant de cet objet, après analyse de la structure de ce composant ;

- une commande de nettoyage ou effacement d'un espace mémoire libéré par un objet structuré, par utilisation du procédé de désérialisation pour créer un objet de contenu sans signification à partir d'une succession linéaire de données déterminée ;

- une commande de duplication d'un objet structuré dit de départ, par utilisation du procédé de sérialisation, sans désallocation de cet objet de

départ, pour créer une succession linéaire de données représentant ce même objet, puis par utilisation du procédé de désérialisation à partir de cette succession linéaire de données pour créer un autre objet structuré de contenu identique à l'objet de départ.

31. Procédé selon la revendication précédente, cecq le langage de programmation de la plate-forme embarquée comporte une première classe (IOApplet) décrivant une méthode abstraite ProcessAPDU, lançant lors de la réception d'un message APDU, un traitement paramétrable dans l'application ; le code programme réalisant les opérations de désérialisation dans la plate-forme embarquée étant mémorisé dans la plate-forme embarquée, en tant qu'implémentation de cette méthode abstraite ProcessAPDU, dans une deuxième classe (ObjectIOApplet) héritant de la première classe (IOApplet), ce même code programme faisant appel à une méthode ProcessObject elle-même décrite comme une méthode abstraite dans cette même classe (ObjectIOApplet) d'implémentation.

32. Procédé selon la revendication (Javacard), cecq le langage de programmation de la plate-forme embarquée comporte une première classe (IOApplet) décrivant une méthode SendAPDU émettant vers l'hôte un message au format APDU ; le code programme réalisant les opérations de sérialisation dans la plate-forme embarquée étant mémorisé, en tant qu'implémentation d'au moins une méthode SendObject faisant appel à la méthode SendAPDU, dans une deuxième classe (ObjectIOApplet) de carte héritant de la classe (IOApplet).

33. Procédé selon l'une des revendications précédentes, caractérisé en ce qu'il est utilisé pour la communication entre au moins un agent logiciel, dit agent de carte, mémorisé ou exécuté dans la plate-forme embarquée et au moins un agent logiciel, dit agent proxy de moteur de carte mémorisé ou exécuté dans au moins un hôte appartenant à un réseau informatique communiquant par messages asynchrones selon une infrastructure logicielle de type AAA-MOM, cet agent proxy de moteur de carte servant d'intermédiaire à

l'agent de carte dans ses communications avec d'autres agents de ce réseau, ces agents fonctionnant selon les spécifications de cette infrastructure logicielle et appartenant à au moins une application distribuée.

34. Système informatique comprenant une station (2) informatique, dite plate-forme embarquée, comprenant un objet portable incluant au moins un processeur, des moyens de mémorisation, et des moyens de communication aptes à échanger des informations avec un terminal sous la forme d'une ou plusieurs successions linéaires de données, caractérisé en ce que la plate-forme comprend un agent (212) de sérialisation apte à effectuer une étape de conversion d'un ensemble de données, dans un sens ou dans l'autre, entre d'une part un agencement en une succession linéaire de données et d'autre part un agencement structuré décrivant ou représentant un ou plusieurs objets logiciels structurés ou hiérarchisés suivant les critères d'un langage de programmation orienté objet.

35. Système selon la revendication précédente, caractérisé en ce que la plate-forme embarquée (2) comprend un agent (211) de communication apte à :

- recevoir en lieu et place de l'agent (221) destinataire un ensemble (32) de données reçues par la fonction (201) de réponse à l'intention d'un agent (221) logiciel destinataire mémorisé dans la plate-forme embarquée, cet ensemble de données étant agencé en une ou plusieurs successions linéaires de données ;

- convertir cet ensemble de données en au moins un objet (34) logiciel, structuré ou hiérarchisé suivant les critères d'un langage de programmation orienté objet ;

- transmettre cet objet (34) logiciel structuré à l'agent (221) destinataire et déclencher un traitement (2210) en fonction de cet objet par cet agent (221) destinataire.

36. Système selon l'une des revendications précédentes, caractérisé en ce que la succession linéaire de données représentant un objet logiciel structuré est mémorisée dans la plate-forme embarquée dans un flux d'entrée ou un flux de sortie, la plate-forme embarquée comportant un agent logiciel dit agent (212) de sérialisation apte à créer dans la plate-forme embarquée le ou les objets structurés représentés par le flux d'entrée, c'est à dire à désérialiser ces objets structurés, ou à écrire dans le flux de sortie des données représentant le ou les objets structurés à transmettre, c'est à dire à sérialiser ces objets structurés.

37. Système selon l'une des revendications précédentes, caractérisé en ce que le flux d'entrée ou le flux de sortie sont mémorisés sous la forme d'une ou plusieurs structures circulaires de mémoire.

38. Système selon l'une des revendications précédentes, caractérisé en ce que l'agent (212) de sérialisation utilise une pile mémoire, dite pile de types, pour mémoriser le type d'au moins un objet composant tout ou partie de la structure d'un objet structuré à sérialiser ou désérialiser, cette pile de type comportant des emplacements mémoires qui ne sont chacun accessibles qu'après que les emplacements mémorisés plus récemment aient été lus et effacés.

39. Système selon l'une des revendications précédentes, caractérisé en ce que les données contenues dans les flux d'entrée ou de sortie représentent un ou plusieurs objets structurés en utilisant un codage comprenant un jeu de balises, ces balises représentant chacune une action déterminée à effectuer lors de la désérialisation de cette succession linéaire de données.

40. Système selon l'une des revendications précédentes, caractérisé en ce qu'au moins une balise est définie comme représentant une des actions suivantes :

- ajout d'un nouvel élément à la structure de l'objet structuré représenté par la succession linéaire de données ;

- référence à un élément ou objet, dit objet source, en tant que source de la valeur de tout ou partie d'un élément composant l'objet structuré ;

- indication que la ou les données suivantes représentent le contenu d'un élément composant l'objet structuré ;

- indication d'une absence de contenu d'un élément composant l'objet structuré.

41. Système selon l'une des revendications précédentes, caractérisé en ce que la plate-forme embarquée comprend un objet portable fonctionnant selon la norme ISO7816 et utilisant des commandes au format APDU.

42. Système selon l'une des revendications précédentes, caractérisé en ce qu'au moins un agent ou application mémorisé dans la plate-forme embarquée est programmé en langage Java®, la plate-forme embarquée comportant un environnement informatique selon le standard JavaCard®.

43. Système selon l'une des revendications précédentes, caractérisé en ce qu'il comporte dans l'hôte ou dans la plate-forme embarquée, ou les deux, au moins une classe logicielle implémentant au moins l'une des commandes suivantes :

- une commande d'écriture d'objet, effectuant la transmission d'un objet (31) structuré à au moins un agent (221) de la carte, par sérialisation de cet objet structuré en un flux de données dans l'hôte, puis envoi de ce flux de données dans la plate-forme embarquée, puis dé-sérialisation de ce flux de données en un objet structuré dans la plate-forme embarquée ;

- une commande de lecture d'objet, effectuant la lecture d'un objet (41) structuré depuis au moins un agent (221) de la carte, par sérialisation de cet objet structuré en un flux de données dans la plate-forme embarquée, puis

réception depuis la plate-forme embarquée de ce flux de données, puis désérialisation de ce flux de données en un objet structuré dans l'hôte ;

- une commande de désallocation d'un objet structuré mémorisé dans la plate-forme embarquée, par une sérialisation de cet objet selon une option comprenant une libération ou désallocation de l'espace mémoire alloué à chaque composant de cet objet, après analyse de la structure de ce composant ;

- une commande de nettoyage ou effacement d'un espace mémoire libéré par un objet structuré dans la plate-forme embarquée, par désérialisation d'une succession linéaire de données déterminée pour créer un objet de contenu sans signification ;

- une commande de duplication dans la plate-forme embarquée d'un objet structuré dit de départ, par sérialisation en une succession linéaire de données représentant ce même objet, sans désallocation de cet objet de départ, puis par désérialisation à partir de cette succession linéaire de données en un autre objet structuré de contenu identique à l'objet de départ.

44. Système selon l'une des revendications précédentes, caractérisé en ce que la plate-forme embarquée communique au moins un hôte appartenant à un réseau informatique communiquant par messages asynchrones selon une infrastructure logicielle de type AAA-MOM, cet hôte comprenant un agent logiciel, dit agent proxy de moteur de carte, apte à gérer les communications de cette plate-forme embarquée avec d'autres agents de ce réseau, ces agents fonctionnant selon les spécifications de cette infrastructure logicielle et appartenant à au moins une application distribuée.

Fig. 1

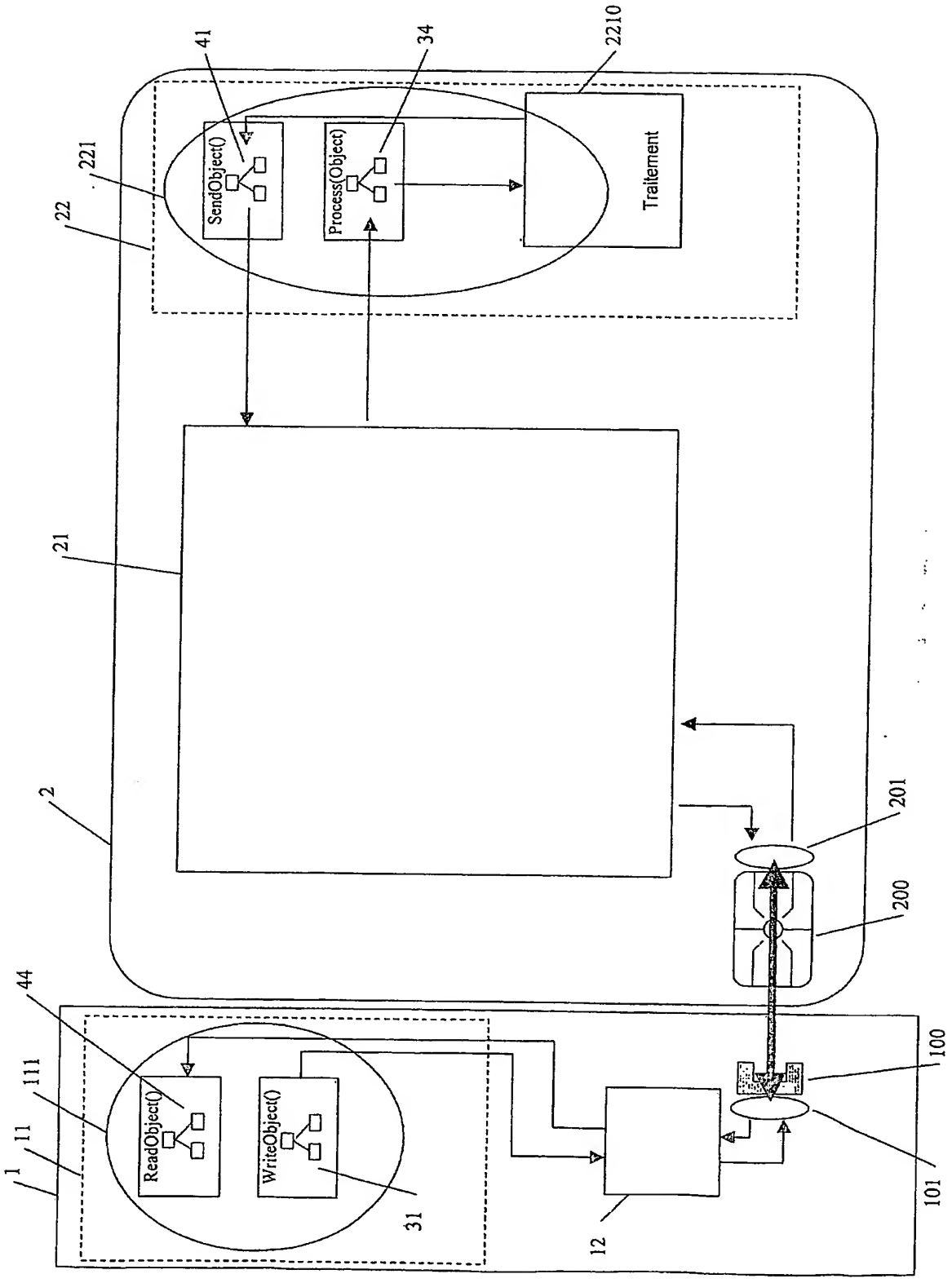

41
34
2210
221
22
SendObject()
Process(Object)
Traitement
21
2
201
200
44
111
11
1
ReadObject()
WriteObject()
31
12
100
101

Fig. 2

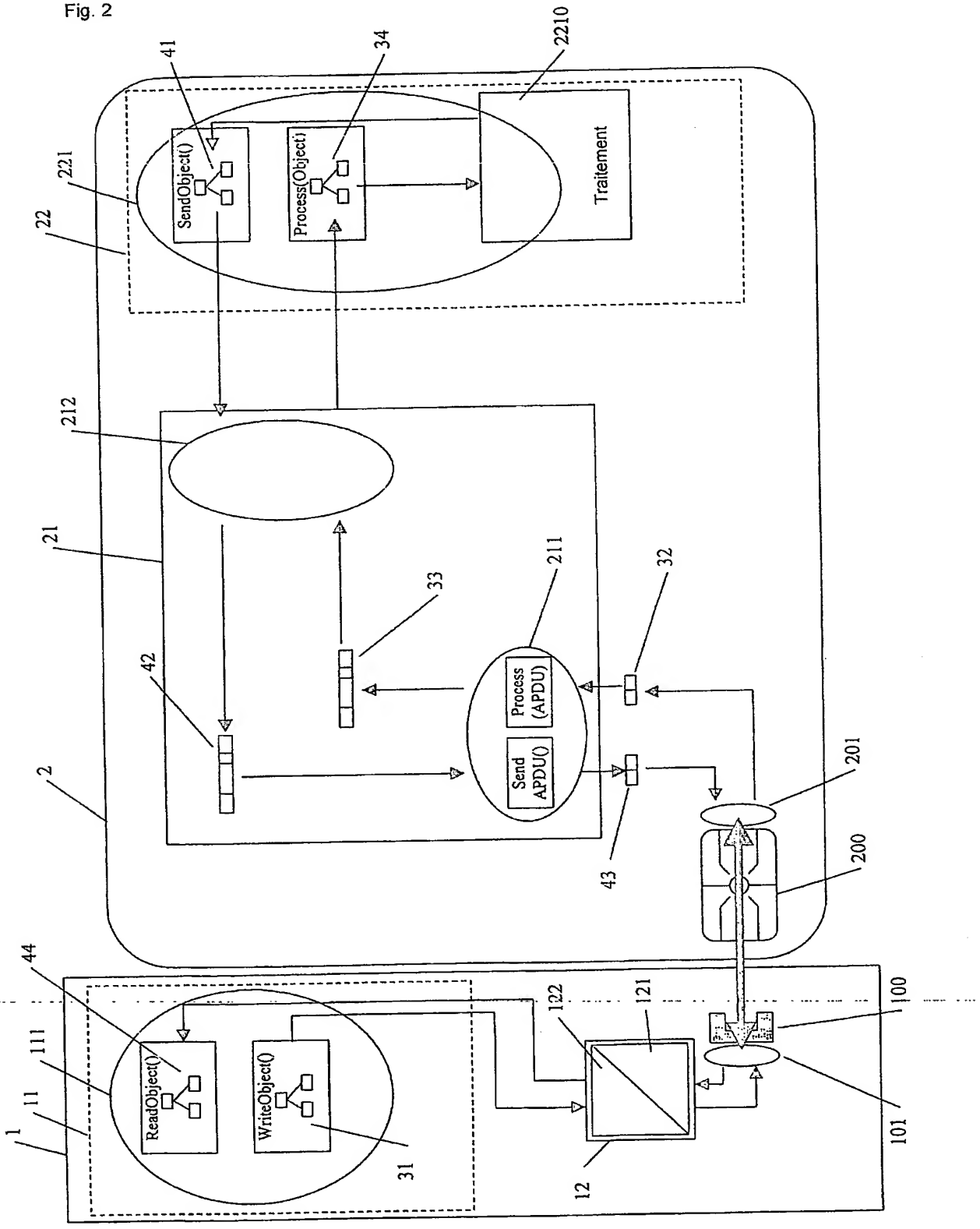

41
34
2210
221
22
SendObject()
Process(Object)
Traitement
212
21
211
33
32
42
Process (APDU)
Send APDU()
2
201
200
43
44
111
11
1
ReadObject()
WriteObject()
31
122
121
100
12
101

Fig. 3

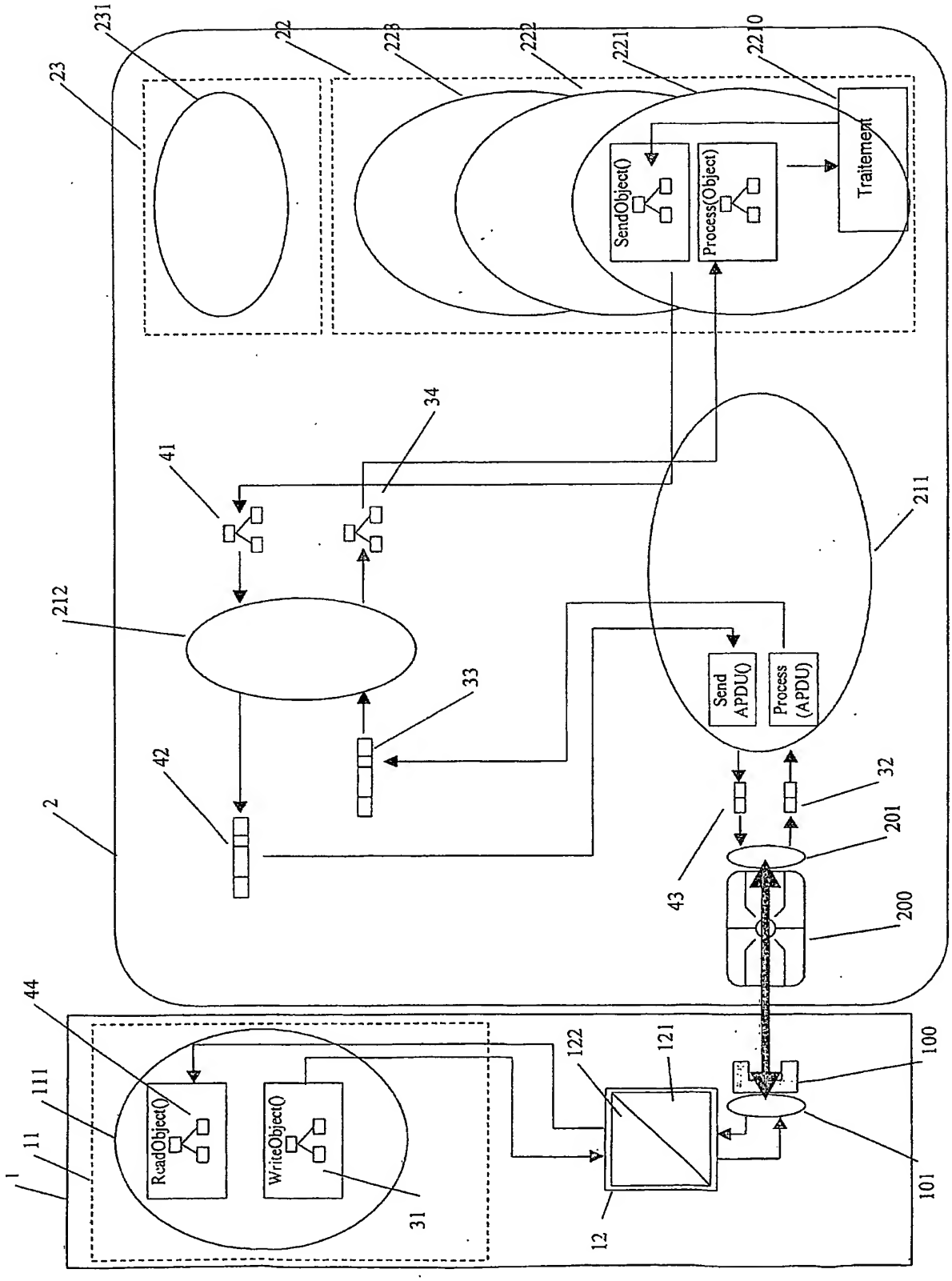

231
22
223
222
221
2210
23
SendObject()
Process(Object)
Traitement
34
41
211
212
Send APDU()
Process (APDU)
33
42
32
2
201
43
200
44
100
111
122
121
11
ReadObject()
WriteObject()
101
1
31
12

Fig. 4

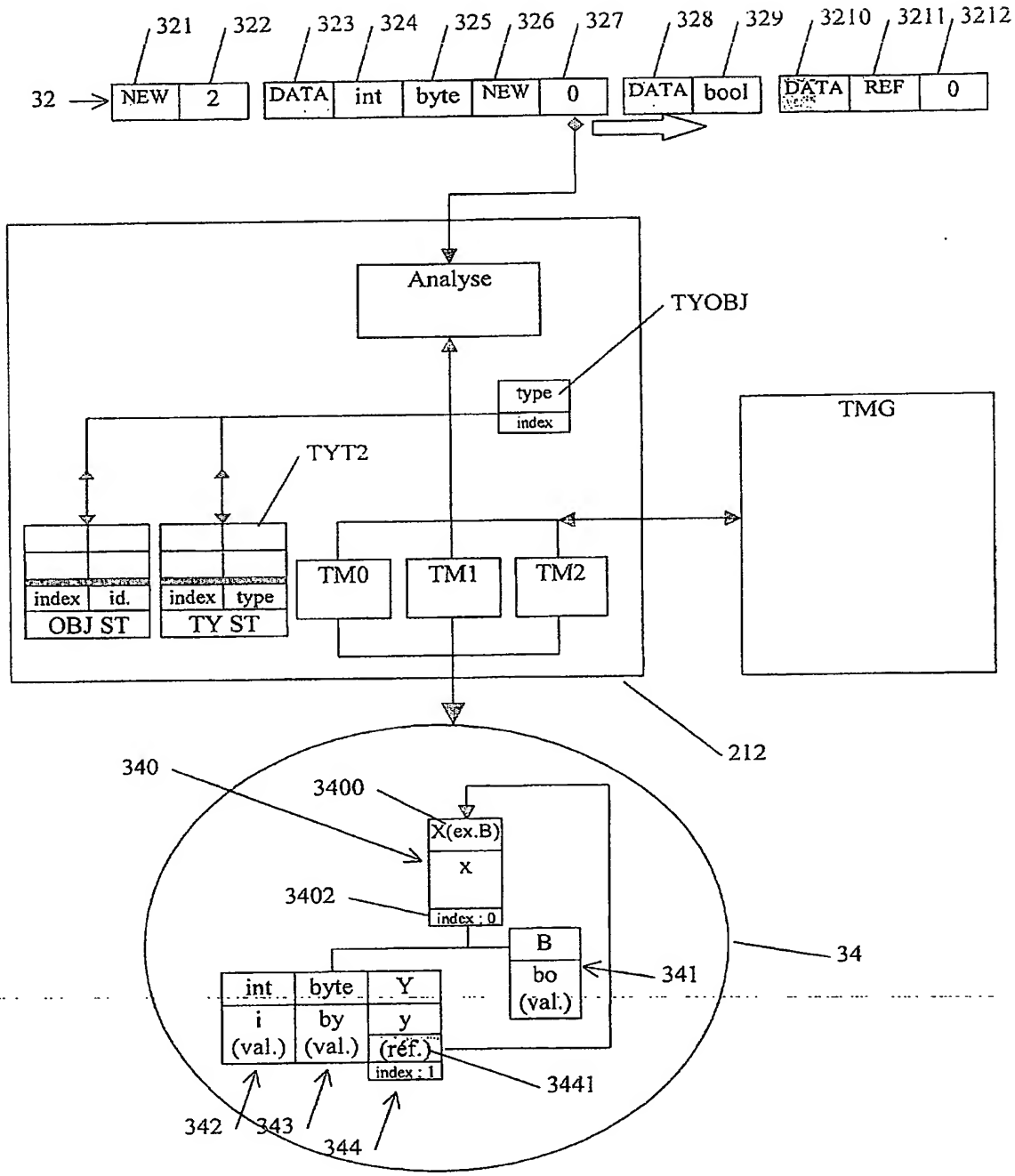
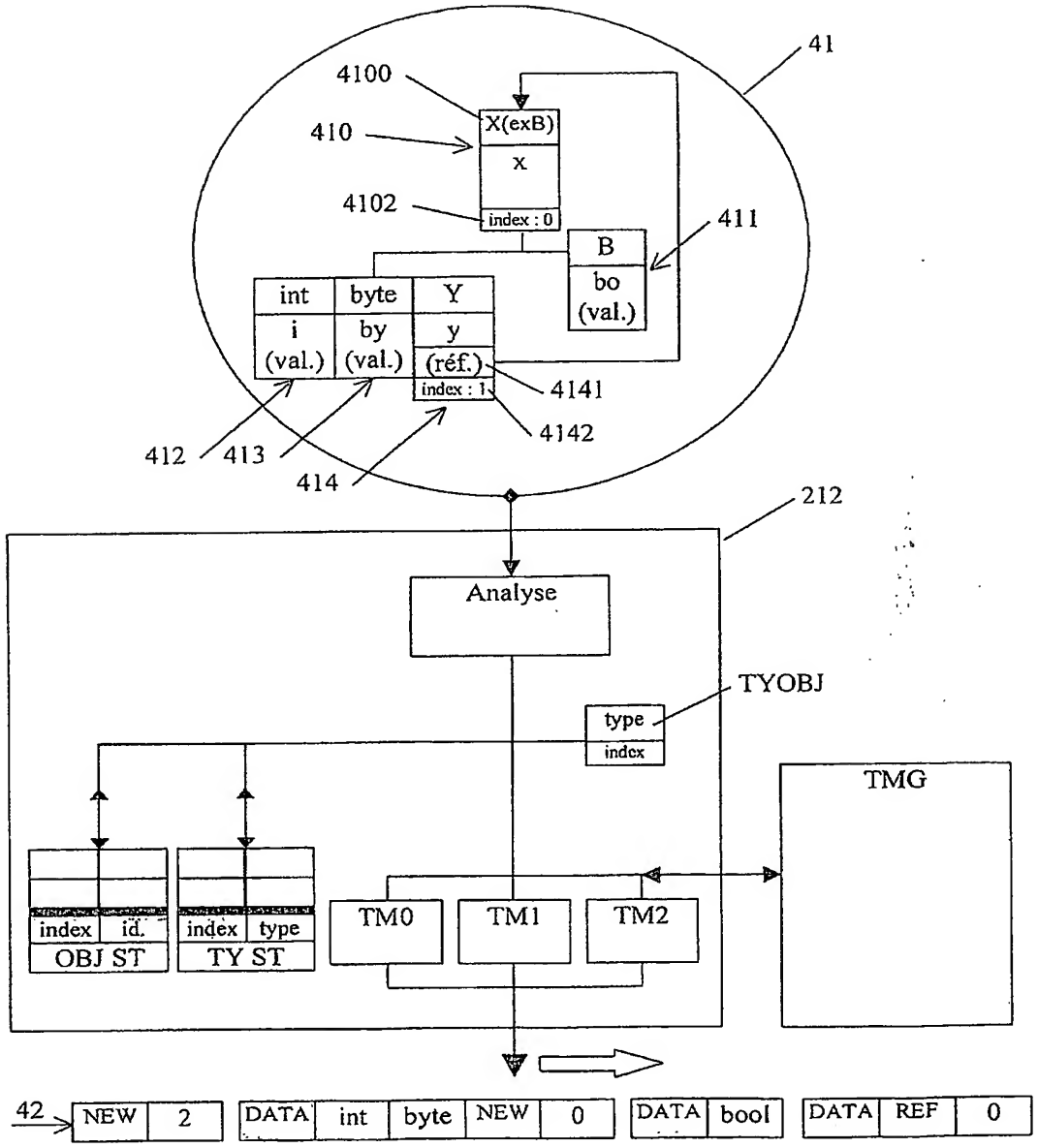

321
322
323
324
325
326
327
328
329
3210
3211
3212
32
NEW
2
DATA
int
byte
NEW
0
DATA
bool
DATA
REF
0
Analyse
TYOBJ
type
index
TMG
TYT2
index
id.
OBJ ST
index
type
TY ST
TM0
TM1
TM2
212
340
3400
X(ex.B)
x
3402
index : 0
B
bo
(val.)
341
34
int
i
(val.)
byte
by
(val.)
Y
y
(ref.)
index : 1
3441
342
343
344

Fig. 5


41
4100
410
X(exB)
x
4102
index : 0
411
B
bo
(val.)
int
byte
Y
i
(val.)
by
(val.)
y
(réf.)
index : 1
4141
4142
412
413
414
212
Analyse
TYOBJ
type
index
TMG
index
id.
OBJ ST
index
type
TY ST
TM0
TM1
TM2
42
NEW
2
DATA
int
byte
NEW
0
DATA
bool
DATA
REF
0

reçue le 09/04/02

INPI
INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE

DÉPARTEMENT DES BREVETS

26 bis, rue de Saint Pétersbourg
75800 Paris Cedex 08
Téléphone : 01 53 04 53 04 Télécopie : 01 42 93 59 30

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI

cerfa N° 11 235*02

DÉSIGNATION D'INVENTEUR(S) Page N° 1../1..
(Si le demandeur n'est pas l'inventeur ou l'unique inventeur)

Cet imprimé est à remplir lisiblement à l'encre noire

DB 113 W /260899

Vos références pour ce dossier *(facultatif)*	BULL 3939 FR
N° D'ENREGISTREMENT NATIONAL	0202570

TITRE DE L'INVENTION (200 caractères ou espaces maximum)
Procédé itératif de sérialisation d'objets logiciels structurés

LE(S) DEMANDEUR(S) :

BULL S.A.
68, route de Versailles
78430 LOUVECIENNES

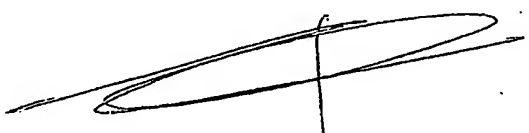
INRIA
Domaine de Voluceau Rocquencourt
B.P.105
78153 LE CHESNAY CEDEX

DESIGNE(NT) EN TANT QU'INVENTEUR(S) : (Indiquez en haut à droite «Page N° 1/1» S'il y a plus de trois inventeurs, utilisez un formulaire identique et numérotez chaque page en indiquant le nombre total de pages).

Nom		FAMBON	
Prénoms		Olivier	
Adresse	Rue	2, rue du commandant Gillot	
	Code postal et ville	38000	GRENOBLE
Société d'appartenance *(facultatif)*			
Nom		FREYSSINET	
Prénoms		André	
Adresse	Rue	Le Sorbier	
	Code postal et ville	38760	SAINT PAUL DE VARCES
Société d'appartenance *(facultatif)*			
Nom		LACOURTE	
Prénoms		Serge	
Adresse	Rue	Le Soleil Levant 2, boulevard des Anciens d'Algérie	
	Code postal et ville	38580	ALLEVARS LES BAINS
Société d'appartenance *(facultatif)*			

DATE ET SIGNATURE(S)
DU (DES) DEMANDEUR(S)
OU DU MANDATAIRE
(Nom et qualité du signataire)

Y. DEBAY Mandataire CPI (92-1066)



La loi n°78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés s'applique aux réponses faites à ce formulaire.
Elle garantit un droit d'accès et de rectification pour les données vous concernant auprès de l'INPI.